Unified 3D Perception and Generative Control for Generalist Robots

Nikolaos Gkanatsios

CMU-RI-TR-25-76

August, 2025

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

Abstract

To enable robot generalists that can operate across tasks, scenes, and embodiments, we need policies that are expressive, multimodal, and grounded in 3D spatial understanding. This thesis explores how unifying 3D perception with generative policy learning advances cross-domain generalization in intelligent robot agents.

We begin by developing foundational 3D perception systems: an open-vocabulary detector adaptable to both 2D and 3D scenes, a unified segmentation model that bridges 2D-3D visual domains, and memory-prompted networks that discover 3D correspondences across scenes without supervision. These advances establish the architectural groundwork for general-purpose 3D manipulation.

Building on these foundations, we present a suite of generative 3D manipulation models—including goal generators, planners, and equivariant policies—that progressively scale in complexity and task versatility. This culminates in 3D Diffuser Actor (3DDA), the first 3D diffusion policy for robotic manipulation, enabling multimodal behaviors and strong task performance. We further generalize this to 3D Flow Actor (3DFA), a versatile policy architecture supporting single-arm, and bimanual manipulation, with 20 times faster training and inference, outperforming even 1000 times larger contemporary models.

Finally, we show that scaling 3D policies to billions of parameters can preserve their spatial grounding and improve data efficiency, when coupled with key architectural choices. This thesis demonstrates that 3D-centric generative policies not only unlock robust and versatile robot behaviors but also pave the way for scalable policy design, advancing the vision of generalist robots.

Acknowledgments

First, I want to thank my advisor, Katerina Fragkiadaki, for being a role model for me as a researcher. During one of our first meetings, Katerina told me that it doesn't matter who you are when you start, it only matters who you become in the process. Even if she values work more than words, I believe she would smile when thinking how much we progressed together.

I also want to thank my committee members: Yonatan Bisk, Shubham Tulsiani and Abhishek Gupta, for all the discussions we've had throughout my studies, research-related or not. Their perspectives and comments had significant impact on this thesis and the PhD journey in general.

Next, I want to thank my family for their continuous support, no matter how far apart we are. From my close family, with whom we have been calling every day, to my extended family, including those I haven't yet met since their recent birth, and those to whom I never had the chance to say a last goodbye.

Lastly, I want to thank everyone else who made this journey remarkable, with special thanks to Xiqiao, Ayush, Tsung-Wei, Gabe, Mayank, Paul, Mohit, Ishita, Yunchu, Andy, Wen-Hsuan, Brian, Mihir, Devon, Archie, Don, Nilay, Moritz, Prachi, Pushkal, Adam, Elfie, Xian, Yash, Matt, Kashu, Alex, Jiahe, Theo, Yuchen and Lei. A shout-out to the broader peer community of Robotics Institute for all the effortless, fruitful and funny, mostly impromptu chats we had on and off campus.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"A robot generalist is a robotic system capable of performing a wide variety of tasks across diverse environments and embodiments, without being explicitly reprogrammed or retrained for each scenario."
ChatGPT, 2025, on "What is a robot generalist"

From robotic arms in factories to autonomous vehicles and household assistants, most robotic systems today are built for narrow tasks in structured settings. They excel at repetition but struggle with adaptation. In contrast, the vision of a robot generalist is one of versatility, where a single system perceives, reasons, and acts across tasks, scenes, and embodiments.

Achieving this vision of a robot generalist remains a central challenge in robotics. Traditionally, robotic systems have been engineered with highly specialized skills, tailored to specific hardware, environments, and task distributions. While effective in narrow domains, these systems lack the flexibility to generalize to new tasks and scenes. Robot learning has emerged as a promising alternative, aiming to enable generalization through experience and data, rather than manual design.

Recent advances in AI have made significant progress by scaling data collection [19, 51, 162, 168, 250, 255, 346, 347], model capacity [15, 253], and co-training with large vision-language models [5, 20, 66]. Yet, these efforts have largely relied on architectures that map 2D images to actions, ignoring the inherently 3D nature of robotic manipulation. This introduces a fundamental mismatch between perception and control. Additionally, while scaling model size improves generalization to a point, it also introduces inefficiencies: these models struggle to adapt to novel tasks with limited demonstrations and often exhibit brittle, viewpoint-dependent behaviors.

Throughout this thesis, we focus on two key principles for building generalist robot policies: explicit 3D spatial grounding and generative modeling of multimodal behaviors. Rather than treating manipulation as a 2D pattern recognition problem, we propose to unify perception and control in a shared 3D representation space. Furthermore, by leveraging generative objectives, policies can model complex, multimodal action distributions, enabling more robust, flexible behaviors in ambiguous and underconstrained tasks.

These principles emerged as recurring motifs across multiple lines of investigation. Starting from perception, we develop architectures that operate jointly in 2D and 3D spaces, leading to improvements in open-vocabulary object detection, segmentation, and cross-scene correspondence. Building on these perception modules, we then introduce a family of generative 3D policies—including goal generators, planners, and actors—that progressively scale in capacity and versatility. Our contributions culminate in the development of 3D Diffuser Actor (3DDA) and 3D Flow Actor (3DFA), which achieve state-of-the-art performance while offering significant efficiency gains over contemporary 2D and diffusion-based methods. We conclude with 3DFA-VLA, an extension of 3DFA which demonstrates the ability of 3D policies to scale and the important design choices for doing so.

The following chapters document this progression, from foundational perception architectures to scalable 3D policies, detailing how these design principles unlock new capabilities in generalist robotic agents.

In Chapters 2, 3, 4, we lay the perception foundations necessary for generalist robotic policies. In Chapter 2, we begin with an open-vocabulary detection model, the first to be capable of achieving state-of-the-art results on both 2D and 3D language grounding benchmarks with minimal adjustments. Building on this, in Chapter 3, we develop a unified 2D-3D segmentation architecture, capable of segmenting entities in both modalities with shared weights, enhancing generalization across viewpoints and sensor types. Finally, in Chapter 4, we introduce memory-prompted segmentation networks, which leverage a retrieval-based mechanism to adapt to novel objects and scenes, facilitating few-shot generalization without the need for weight updates. These perception models serve as critical bottlenecks that abstract the scene into task-relevant representations.

In Chapters 5, 6 and 7, we transition to policy learning, progressively developing a family of generative 3D manipulation policies. In Chapter 5, we begin by factorizing policy learning into modular components: high-level language-conditioned planning, scene abstraction through perception bottlenecks, goal configuration planning, and low-level action prediction. This modular design enables generalization to unseen domains and real-world tasks without fine-tuning. Next, in Chapter 6, we propose Act3D, an equivariant architecture that predicts goal keyposes in 3D space, achieving strong generalization across a large suite of manipulation tasks. However, we observe limitations in handling tasks requiring fine-grained continuous interaction due to reliance on heuristic planners. To address this, in Chapter 7, we introduce diffusion-based planners, reframing trajectory generation as a denoising problem. This replacement significantly improves policy expressivity and performance in complex manipulation tasks involving articulated and deformable objects.

In Chapter 8, we unify these insights into 3D Diffuser Actor (3DDA), the first end-to-end diffusion-based 3D manipulation policy. 3DDA models multimodal action distributions while retaining explicit 3D spatial grounding, achieving strong performance across diverse benchmarks. While 3DDA establishes a new state-of-the-art, its scalability is constrained by its prolonged training times and slow inference.

To overcome these limitations, in Chapter 9, we introduce 3D Flow Actor (3DFA), a versatile policy that generalizes the 3DDA framework with a continuous-time flow model. 3DFA enables up to 20 times faster training and inference while supporting single-arm, bimanual and dexterous manipulation.

Finally, in Chapter 10 we conclude with future work on an extension of 3DFA, called 3DFA-VLA, a scalable variant with one billion parameters, demonstrating how 3D policies can scale efficiently while preserving 3D token grounding and outperforming vision-language-action (VLA) baselines in terms of data efficiency and generalization.

This thesis advances the goal of building generalist robot policies by unifying 3D spatial grounding with generative modeling of multimodal behaviors. The key contributions are as follows:

1. Unified 2D-3D Perception Architectures: We introduce novel architectures that operate seamlessly across 2D and 3D representations, including:

   - An open-vocabulary detector capable of grounding language queries in 2D and 3D scenes with minimal architectural changes.
   - A joint segmentation network that improves performance on both 2D and 3D inputs through shared representations.
   - Memory-prompted segmentation models that exhibit emergent capabilities such as unsupervised 3D correspondence discovery and rapid adaptation to new object categories.

2. Generative 3D Manipulation Policies: We develop a family of policies that leverage generative models to handle the inherent multimodality of manipulation tasks:
   - An energy-based goal generator for spatial rearrangement.
   - A goal-conditioned planner based on diffusion models that enables flexible trajectory prediction.
   - 3D Diffuser Actor (3DDA), the first diffusion policy operating directly in 3D spaces, achieving state-of-the-art performance across multiple manipulation benchmarks.

3. Scalable and Efficient 3D Generative Policy Architectures: We propose architectural innovations that enable 3D policies to scale in parameter size while maintaining explicit 3D token grounding:
   - 3D Flow Actor (3DFA), a versatile policy architecture supporting single-arm and bimanual manipulation, with significant efficiency improvements over existing diffusion methods.
   - 3DFA-VLA, a large-scale extension demonstrating that 3D policies can match or surpass 2D VLM-based models in performance and data efficiency, even as model capacity and data increases.

Chapter 2

Bottom Up Top Down Detection Transformers for Language Grounding in Images and Point Clouds

2.1 Introduction

Language-directed attention helps us localize objects that our "bottom-up", task-agnostic perception may miss. Consider Fig. 2.1. The utterance *"bottle on top of the bathroom vanity"* suffices to direct our attention to the reference object, even though it is far from salient. Language-directed perception adapts the visual processing of the input scene according to the utterance. Object detectors instead apply the same computation in each scene, which can miss task-relevant objects.

Most existing language grounding models use object proposal bottlenecks: they select the referenced object from a pool of object proposals provided by the pre-trained object detector [74, 85, 119, 151, 161]. This means they cannot recover objects or parts that a bottom-up detector misses. This is limiting since small, occluded, or rare objects are hard to detect without task-driven guidance. For example, in Figure 2.1

This chapter is based on the paper previously published at ECCV 2022 [135]

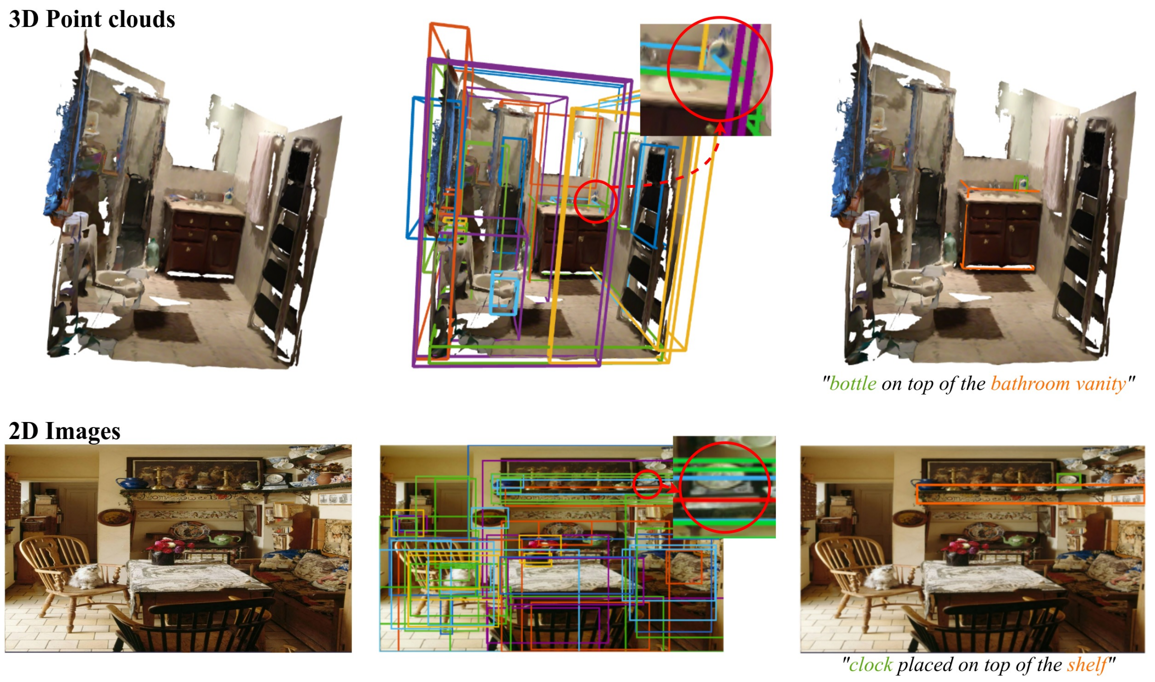

3D Point clouds
"bottle on top of the bathroom vanity"
2D Images
"clock placed on top of the shelf"

Figure 2.1: **Language-modulated 3D (*top*) and 2D (*bottom*) detection with BUTD-DETR.** *Middle:* State-of-the-art object detectors often fail to localize small, occluded or rare objects (here they miss the clock on the shelf and the bottle on the cabinet). *Right:* Language-driven and objectness-driven attention in BUTD-DETR modulates the visual processing depending on the referential expression while taking into account salient, bottom-up detected objects, and correctly localizes all referenced objects.

middle, state-of-the-art 2D [285] and 3D [216] detectors miss the clock on the shelf and the bottle on the bathroom vanity, respectively.

Recently, Kamath et al. [158] introduced MDETR, a language grounding model for 2D images that decodes object boxes using a DETR [23] detection head and aligns them to the relevant spans in the input utterance, it does not select the answer from a box proposal pool. The visual computation is modulated based on the input utterance through several layers of self-attention on a concatenation of language and visual features. MDETR achieves big leaps in performance in 2D language grounding over previous box-bottlenecked methods.

We propose a model for grounding referential utterances in 3D and 2D visual scenes that builds upon MDETR, which we call BUTD-DETR (pronounced Beauty-

DETR), as it uses both box proposals, obtained by a pre-trained detector "bottom-up" and "top-down" guidance from the language utterance, to localize the relevant objects in the scene. BUTD-DETR uses box proposals obtained by a pre-trained detector as an additional input stream to attend on; however, it is not box-bottlenecked and still decodes objects with a detection head, instead of selecting them from the input box stream. Current object detectors provide a noisy tokenization of the input visual scene that, as our experiments show, is a useful cue to attend on for multimodal reasoning. Second, BUTD-DETR augments grounding annotations by configuring annotations for object detection as detection prompts to be grounded in visual scenes. A detection prompt is a list of object category labels, e.g., *"Chair. Door. Person. Bed."*. We train the model to ground detection prompts by localizing the labels that are present in the image and learn to discard labels that are mentioned but do not correspond to any objects in the scene. Third, BUTD-DETR considers improved bounding box - word span alignment losses that reduce noise during alignment of object boxes to noun phrases in the referential utterance.

We test BUTD-DETR on the 3D benchmarks of [2, 25] and 2D benchmarks of [163, 388]. In 3D point clouds, we set new state-of-the-art in the two benchmarks of Referit3D [2] and ScanRefer [25] and report significant performance boosts over all prior methods (12.6% on SR3D, 11.6% on NR3D and 6.3% on ScanRefer), as well as over a direct MDETR-3D implementation of ours that does not use a box proposal stream or detection prompts during training. In 2D images, our model obtains competitive performance with MDETR on RefCOCO, RefCOCO+ and Flickr30k, and requires less than half of the GPU training time due to the cheaper deformable attention in the visual stream. We ablate each of the design choices of the model to quantify their contribution to performance.

In summary, our contributions are: **(i)** A model with SOTA performance across both 2D and 3D scenes with minor changes showing that modulated detection in 2D images can also work in 3D point clouds with appropriate visual encoder and decoder modifications. **(ii)** Augmenting supervision with detection prompts, attention on an additional input box stream and improved bounding box - word span alignment losses. **(iii)** Extensive ablations to quantify the contribution of different components of our model. We make our code publicly available at https://butd-detr.github.io.

2.2 Related work

Object detection with transformers

Object detectors are trained to localize all instances of a closed set of object category labels in images and 3D point-clouds. While earlier architectures pool features within proposed boxes to decode objects and classify them into categories [114, 205, 283], recent methods pioneered by DETR [23] use transformer architectures where a set of object query vectors attend to the scene and among themselves to decode object boxes and their labels. DETR suffers from the quadratic cost of within image features self attention. D(eformable)-DETR [412] proposes deformable attention, a locally adaptive kernel that is predicted directly in each pixel location without attention to other pixel locations, thus saving the quadratic cost of pixel-to-pixel attention. Our model builds upon deformable attention for feature extraction from RGB images. [216, 242] extend detection transformers to 3D point cloud input.

2D referential language grounding

Referential language grounding [163] is the task of localizing the object(s) referenced in a language utterance. Most 2D language grounding models obtain sets of object proposals using pre-trained object detectors and the original image is discarded upon extraction of the object proposals [74, 85, 119, 151, 161]. Many of these approaches use multiple layers of attention to fuse information across both, the extracted boxes and language utterance [33, 222, 381]. Recently, a few approaches directly regress the target bounding box without using pre-trained object proposals. In [32] language and visual features cross-attend and are concatenated to predict the box of the referential object. Yang et al. [379] extends the YOLO detector [283] to referential grounding by channel-wise concatenating language, visual and spatial feature maps and then regressing a single box using the YOLO box prediction head. [297] performs a fusion similar to [379], then selects a single box from a set of anchor boxes and predicts a deformation of it, much like the Faster-RCNN object detector [285]. While previous approaches encode the whole text input into a single feature vector, [380] further improves performance by recursively attending on different parts of the referential utterance. Lastly, [56] encodes the image and utterance with within- and cross-modality transformers, and a special learnable token regresses a single box. In contrast to our method, all these works predict a single bounding box per image-utterance pair. Our work builds upon MDETR of Kamath et al. [158] that modulates visual processing through attention to the input language utterance and decodes objects from queries similar to DETR, without selecting from a pool of proposals. Both our method and MDETR can predict multiple instances being referred to, as well as ground intermediate noun phrases. Concurrent to our work, GLIP [182] shows that adding supervision from detection annotations can improve 2D referential grounding. Our work independently confirms this hypothesis in 2D and also shows its applicability on the 3D domain.

3D referential language grounding

has only recently gained popularity [2, 25]. To the best of our knowledge, all related approaches are box-bottlenecked: they extract 3D object proposals and select one as their answer. Their pipeline can be decomposed into three main steps: i) Representation of object boxes as point features [381], segmentation masks [390] or pure spatial/categorical features [290]. ii) Encoding of language utterance using word embeddings [290, 381] and/or scene graphs [77]. iii) Fusion of the two modalities and scoring of each proposal using graph networks [124] or Transformers [381]. Most of these works also employ domain-specific design choices by explicitly encoding pairwise relationships [109, 124, 390] or by relying on heuristics, such as restricting attention to be local [390, 404] and ignoring input modalities [290]. Such design prevents those architectures from being applicable to both the 3D and 2D domains simultaneously.

Due to the inferior performance of 3D object detectors in comparison to their 2D counterparts, popular benchmarks for 3D language grounding, such as Referit3D [2] provide access to ground-truth object boxes at test time. The proposed BUTD-DETR is the first 3D language grounding model that is evaluated on this benchmark without access to oracle 3D object boxes.

2.3 Method

We first describe MDETR [158] in Section 2.3.1. Then, we present BUTD-DETR's architecture in Section 2.3.2, supervision augmentation with detection prompts in Section 2.3.3 and its training objectives in Section 2.3.4.

2.3.1 Background: MDETR

MDETR is a 2D language grounding model that takes a referential utterance and an RGB image as input and localises in the image all objects mentioned in the utterance. MDETR encodes the image with a convolutional network [111] and the language utterance with a RoBERTa encoder [211]. It then fuses information across the language and visual features through multiple layers of self-attention on the concatenated visual and language feature sequences. In MDETR's decoder, a set of query vectors iteratively attend to the contextualized visual features and self-attend to one another, similar to the DETR's [23] decoder. Finally, each query decodes a bounding box and a confidence score over each word in the input utterance, which associates the box to a text span.

The predicted boxes are assigned to ground-truth ones using a Hungarian matching, similar to [23]. Upon matching, the following losses are computed:

- A bounding box loss between predicted boxes and the corresponding ground-truth ones. This is a combination of L1 and generalized IoU [288] losses.
- A soft token prediction loss. A query matched to a ground-truth box is trained to decode a uniform distribution over the language token positions that refer to that object. Queries not matched to ground-truth targets are trained to predict a no-object label.
- Two contrastive losses between query and language token features. The first one, called *object contrastive loss*, pulls an object query's features closer to the features of the corresponding ground-truth span's word tokens, and further than all other tokens. The second one, called *token contrastive loss*, pulls the features of a ground-truth span's token closer to the corresponding object query features, and further than all other queries.

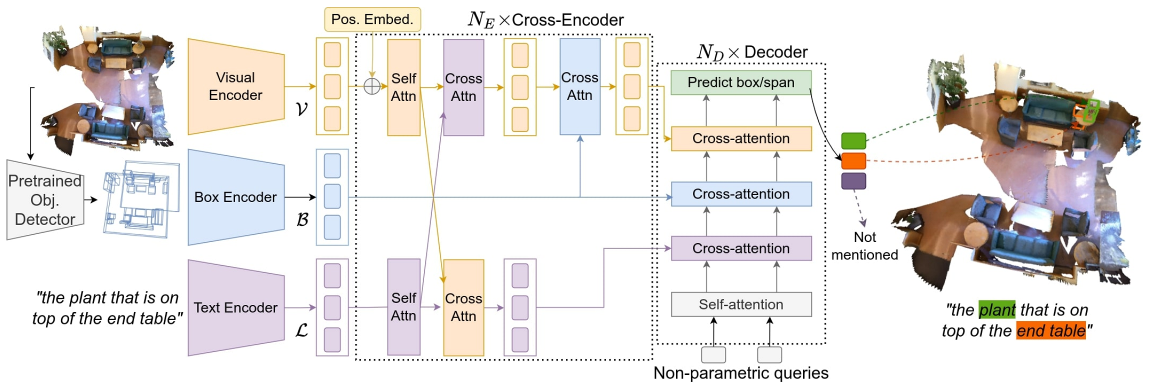


Figure 2.2: **BUTD-DETR architecture.** Given a visual scene and a referential utterance, the model localizes all object instances mentioned in the utterance. A pre-trained object detector extracts object box proposals. The visual scene features, the language utterance and the labelled box proposals are encoded into corresponding sequences of visual, word and box tokens using visual, language and box encoders, respectively. The three streams cross-attend and finally decode boxes and corresponding spans in the language utterance that each decoded box refers to. We visualize here the model operating on a 3D point cloud; an analogous architecture is used for 2D image grounding.

2.3.2 Bottom-up Top-down DETR (BUTD-DETR)

The architecture of BUTD-DETR is illustrated in Figure 2.2. Given a referential language utterance, e.g., “find the plant that is on top of the end table” and a visual scene, which can be a 3D point cloud or a 2D image, BUTD-DETR is trained to localize all objects mentioned in the utterance. In the previous example, we expect one box for the “plant” and one for the “end table”. The model attends across image/point cloud, language and box proposal streams, then decodes the relevant objects and aligns them to input language spans.

Within-modality encoder

In 2D, we encode an RGB image using a pre-trained ResNet101 backbone [110]. The 2D appearance visual features are added to 2D Fourier positional encodings, same as in [132, 412]. In 3D, we encode a 3D point cloud using a PointNet++ backbone [270]. The 3D point visual features are added to learnable 3D positional encodings, same as in [216]: we pass the coordinates of the points through a small multilayer perceptron (MLP). Let $\mathcal{V} \in \mathbb{R}^{n_v \times c_v}$ denote the visual token sequence, where n_v is the number of visual tokens and c_v is the number of visual feature channels.

The words of the input utterance are encoded using a pre-trained RoBERTa [211] backbone. Let $\mathcal{L} \in \mathbb{R}^{n_\ell \times c_\ell}$ denote the word token sequence.

A pre-trained detector is used to obtain 2D or 3D object box proposals. Following prior literature, we use Faster-RCNN [285] for RGB images, pre-trained on 1601 object categories of Visual Genome [175], and Group-Free detector [216] for 3D point clouds pre-trained on a vocabulary of 485 object categories on ScanNet [49]. The detected box proposals that surpass a confidence threshold are encoded using a box proposal encoder, by mapping their spatial coordinates and categorical class information to an embedding vector each, and concatenating them to form an object proposal token. We use a pre-trained and frozen RoBERTa [211] backbone to encode the semantic categories of proposed boxes. Let $\mathcal{O} \in \mathbb{R}^{n_o \times c_o}$ denote the object token sequence.

The 3D detector is trained on ScanNet and all 3D benchmarks we use are also ScanNet-based. This creates a discrepancy in the quality of the detector's predictions between train and test time, as it is far more accurate on the training set. As a result, we find that BUTD-DETR tends to rely on the detector at training time and generalizes less at test time, where the detector's predictions are much noisier. To mitigate this, we randomly replace 30% of the detected boxes at training time with random ones. This augmentation leads to stronger generalization when the detector fails to locate the target object. Note that this is not the case in 2D, where the detector is trained on a different dataset.

All visual, word and box proposal tokens are mapped using (different per modality) MLPs to same-length feature vectors.

Cross-modality Encoder

The visual, language and box proposals, interact through a sequence of N_E cross-attention layers. In each encoding layer, visual and language tokens cross-attend to one another and are updated using standard key-value attention. Then, the resulting language-conditioned visual tokens attend to the box proposal tokens. We use standard attention for both streams in 3D and deformable attention [412] for the visual stream in 2D.

In contrast to MDETR, BUTD-DETR keeps visual, language and box stream separate in the encoder instead of concatenating them. This enables us to employ deformable attention [412] in self and cross attention layers involving the visual stream in 2D domain. Deformable attention involves computing bilinearly interpolated features which is expensive and non-robust in discontinous and sparse modalities like pointclouds, hence we use vanilla attention in 3D. In our experiments, we show that concatenation versus keeping separate streams performs similarly in 3D referential grounding.

Decoder

BUTD-DETR decodes objects from contextualized features using non-parametric queries in both 2D and 3D, similar to [216, 412]. Non-parametric queries are predicted by visual tokens from the current scene, in contrast to parametric queries used in DETR [23] and MDETR [158] that correspond to a learned set of vectors shared across all scenes. Specifically, the contextualized visual tokens from the last multi-modality encoding layer predict confidence scores, one per visual token. The top-K highest scoring tokens are each fed into an MLP to predict a vector which stands for an *object query*, i.e., a vector that will decode a box center and size relative to the location of the corresponding visual token, similar to D-DETR [412]. The query vectors are updated in a residual manner through N_D decoder layers. In each decoder layer, we employ four types of attention operations. First, the queries self-attend to one another to contextually refine their estimates. Second, they attend to the contextualized word embeddings to condition on the language utterance. Next, they attend to the box proposal tokens and then in the image or point visual tokens. At the end of each decoding layer, there is a prediction head that predicts a box center displacement, height and width vector, and a token span for each object query that localizes the corresponding object box and aligns it with the language input.

2.3.3 Augmenting supervision with detection prompts

Object detection is an instance of referential language grounding in which the utterance is a single word, namely, the object category label. Language grounding models have effectively combined supervision across referential grounding, caption description and

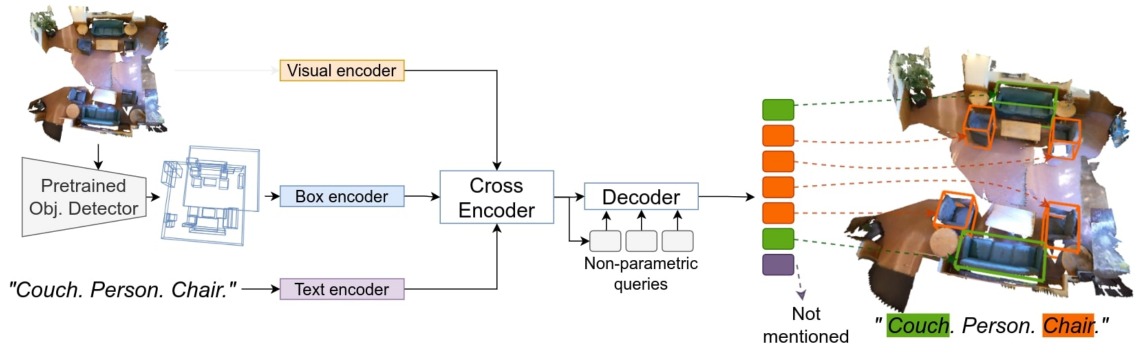


Figure 2.3: **Augmenting referential grounding supervision with detection prompts.** A detection prompt is constructed by sequencing sampled object category labels (here *couch*, *person* and *chair*). The task is to localize all instances of mentioned objects and associate them with the correct span in the prompt. 50% of the sampled labels are negative, i.e., they have no corresponding object instance in the scene. The model learns not to associate these spans with predicted boxes.

question answering tasks [222, 223], which is an important factor for their success. Object detection annotations have not been considered so far as candidates for such co-training.

We cast object detection as grounding of detection prompts, namely, referential utterances comprised of a list of object category labels, as shown in Figure 2.3. Specifically, given the detector's vocabulary of object category labels, we randomly sample a fixed number of them—some appear in the visual scene and some do not—and generate synthetic utterances by sequencing the sampled labels, e.g., *"Couch. Person. Chair. Fridge."*, we call them detection prompts. We treat these prompts as referential utterances to be grounded: the task is to localize *all* object instances of the category labels mentioned in the prompt if they appear in the scene. The sampling of negative category labels (labels for which there are no object instances present) operates as negative training: the model is trained to not match any boxes to the negative category labels.

2.3.4 Supervision objectives

We supervise the outputs of all prediction heads in each layer of the decoder. We follow MDETR [158] in using Hungarian matching to assign a subset of object queries to the ground-truth object boxes and then compute the bounding box, soft token prediction and contrastive losses. Our bounding box and soft token prediction losses are identical to MDETR's. However, we notice that MDETR's contrastive losses do not compare all object queries and word tokens symmetrically. Specifically, the object contrastive loss supervises only the object queries that are matched to a ground-truth object box. On the other hand, the token contrastive loss includes only the tokens that belong to positive spans, namely, noun phrases with corresponding object instances in the scene. As a result, object queries not matched to any ground-truth object box are not pulled far from non-ground-truth text spans, which means at inference object queries can be close to negative spans. We find this asymmetry to hurt performance, as we show in our experiments.

To address this, we propose a symmetric alternative where the similarities between all object queries and language tokens are considered. We append the span "not-mentioned" to all input utterances. This acts as the ground-truth text span for all object queries that are not assigned to any of the ground-truth objects. The object contrastive loss now supervises all queries and considers the similarities with all tokens. We empirically find that gathering unmatched queries to "not mentioned" is beneficial. This is similar in principle to the soft token prediction loss, where unmatched queries have to predict "no object". In fact, we find that this symmetric contrastive loss is sufficient for our model's supervision, but we observe that co-optimizing for soft token prediction results in faster convergence.

2.4 Experiments

We test BUTD-DETR on grounding referential utterances in 3D point clouds and 2D images. Our experiments aim to answer the following questions:

1. How does BUTD-DETR perform compared to the state-of-the-art in 3D and 2D language grounding?
2. How does BUTD-DETR perform compared to a straightforward extension of the 2D state-of-the-art MDETR [158] model in 3D?
3. How much, if at all, attending to a bottom-up box proposal stream helps performance?

4. How much, if at all, co-training for grounding detection prompts helps performance?
5. How much, if at all, the proposed contrastive loss variant helps performance?

2.4.1 Language grounding in 3D point clouds

We test BUTD-DETR on SR3D, NR3D [2] and ScanRefer [25] benchmarks. All three benchmarks contain pairs of 3D point clouds of indoor scenes from ScanNet [49] and corresponding referential utterances, and the task is to localize the objects referenced in the utterance. The utterances in SR3D are short and synthetic, e.g., *"choose the couch that is underneath the picture"*, while utterances in NR3D and ScanRefer are longer and more natural, e.g. *"from the set of chairs against the wall, the chair farthest from the red wall, in the group of chairs that is closer to the red wall"*. For fair comparison against previous methods, we train BUTD-DETR separately on each of SR3D, NR3D and ScanRefer. We augment supervision in each of the three datasets with ScanNet detection prompts. SR3D provides annotations for all objects mentioned in the utterance, so during training we supervise localization of all objects mentioned. In NR3D and ScanRefer, we use supervision for grounding only the referenced object.

All existing models that have been tested in SR3D or NR3D benchmarks are box-bottlenecked, namely, they are trained to select the answer from a pool of box proposals. They all use **ground-truth 3D object boxes (without category labels)** as the set of boxes to select from. We thus consider two evaluation setups:

1. `det`: where we re-train previous models using their publicly available code and provide the same 3D box proposals we use in BUTD-DETR, obtained by the Group-Free 3D object detector [216] trained to detect 485 object categories in ScanNet (Section `det` in Table 2.1).
2. `GT`, where we use ground-truth 3D object boxes for our model and baseline (Section `GT` in Table 2.1).

Alongside previous models, we also compare our model against our implementation of the MDETR model in 3D. This is similar to our model but without attention on a box stream, without co-training with detection prompts and with the original contrastive losses proposed by MDETR. We also replace MDETR's parametric object

Table 2.1: **Results on language grounding in 3D point clouds.** We evaluate top-1 accuracy using ground-truth (`GT`) or detected (`det`) boxes. * denotes method uses extra 2D image features. † denotes evaluation with detected boxes using the authors' code and checkpoints. ‡ denotes re-training using the authors' code. For [404], we compare against their 3D-only version.

	SR3D		NR3D	ScanRefer (Val. Set)	
Method	Acc@0.25(`det`)	Acc.(`GT`)	Acc@0.25(`det`)	Acc@0.25(`det`)	Acc@0.5(`det`)
ReferIt3DNet [2]	$27.7^\dagger$	39.8	$24.0^\dagger$	26.4	16.9
ScanRefer [25]	-	-	-	35.5	22.4
TGNN [124]	-	45.0	-	37.4	29.7
3DRefTransformer [1]	-	47.0	-	-	-
InstanceRefer [390]	$31.5^\ddagger$	48.0	$29.9^\ddagger$	40.2	32.9
FFL-3DOG [77]	-	-	-	41.3	34.0
LanguageRefer [290]	$\underline{39.5}^\dagger$	56.0	$28.6^\dagger$	-	-
3DVG-Transformer [404]	-	51.4	-	<u>45.9</u>	<u>34.5</u>
TransRefer3D [109]	-	57.4	-	-	-
SAT-2D [381]*	$35.4^\dagger$	<u>57.9</u>	$\underline{31.7}^\dagger$	44.5	30.1
MDETR-[158]-3D (our impl.)	45.4	-	31.5	47.2	31.9
BUTD-DETR (ours)	**52.1**	**67.0**	**43.3**	**52.2**	**39.8**

queries with non-parametric one —similar to our model—since they have been shown to be crucial for good performance in 3D [216, 242]. We call this model MDETR-3D. For the sake of completeness, we do have a 3D version of MDETR that uses parametric queries in Table 2.2 and, as expected, it is significantly worse. MDETR does not use a pool of box proposals in any way and hence we cannot report results of MDETR-3D under `GT`.

We show quantitative results of our models against previous works in Table 2.1. We use top-1 accuracy metric, which measures the percentage of times we can find the target box with an IoU higher than the threshold. We report results with IoU@0.25 on SR3D and NR3D; and with both IoU@0.25 and IoU@0.5 on ScanRefer. Please refer to supplementary for more detailed results.

BUTD-DETR outperforms existing approaches as well as MDETR-3D by a large margin under both evaluation setups, `det` and `GT`. It also outperforms the recent SAT-2D [381] that uses additional 2D RGB image features during training. BUTD-DETR does not use 2D image features, but it can be easily extended to do so. We show qualitative results in Figure 2.4.

Table 2.2: **Ablation of design choices for BUTD-DETR on SR3D**.

Model	Accuracy
BUTD-DETR	**52.1**
w/o visual tokens	41.9
w/o detection prompts	47.9
w/o box stream	51.0
with MDETR's [158] contrastive loss	49.6
w/o detection prompts; w/o box stream; (MDETR [158]-3D)	45.4
with parametric queries; w/o detection prompts; w/o box stream; (MDETR [158]-3D-Param)	33.8
with concatenated Visual, Language and Object Streams	51.3

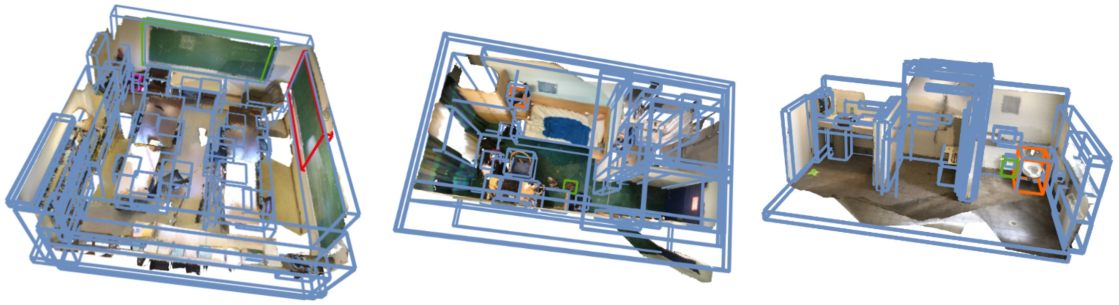
Ablative analysis

We ablate all our design choices for 3D BUTD-DETR on SR3D benchmark [2] in Table 2.2. We compare BUTD-DETR against the following variants:

- w/o visual tokens: an object-bottlenecked variant, which only attends to the language and box proposal streams and selects one box out of the proposals.
- w/o detection prompts: BUTD-DETR trained solely on SR3D grounding utterances.
- w/o box stream: BUTD-DETR without attention on the box stream.
- w/ MDETR's contrastive loss: BUTD-DETR where we replace our modified contrastive loss with MDETR's.
- w/o detection prompts, w/o box stream, w/ MDETR's contrastive loss: an MDETR [158]-3D implementation.
- w/ parametric queries, w/o detection prompts, w/o box stream, w/ MDETR's contrastive loss: an MDETR-3D implementation that uses parametric object queries, as in original MDETR.
- w/ concatenated visual, language and box streams: instead of attending to each modality separately, we concatenate the different streams along their sequence dimension.

The conclusions are as follows:

1. **Box bottlenecks hurt:** Models such as BUTD-DETR and MDETR-3D that decode object boxes instead of selecting them from a pool of given object proposals significantly outperform box-bottlenecked variants. BUTD-DETR



"facing the front of the trash can, pick the blackboard that is to the right of it"
(a)

"find the shoes in front of the tv"
(b)

"select the dustbin next to toilet"
(c)

Figure 2.4: **Qualitative results of BUTD-DETR in the SR3D benchmark**. Predictions for the target are shown in green and for other mentioned objects in orange. The detected proposals appear in blue. (a) The variant without box stream (red box) fails to exploit the information given by the detector, but BUTD-DETR succeeds. (b) The detector misses the "shoes" and any box-bottlenecked variant fails. (c) The detector is successful in finding the "dustbin", still BUTD-DETR refines the box to get a more accurate bounding box.

outperforms by 10.2% an object-bottlenecked variant, that does not attend to 3D point features and does not decode boxes.

2. **BUTD-DETR outperforms MDETR-3D by 6.7%**:
3. **Attention on a box proposal stream helps:** Removing attention on the box stream causes an absolute 1.1% drop in accuracy.
4. **Co-training with detection prompts helps:** Co-training with detection prompts contributes 4.2% in performance (from 47.9% to 52.1%).
5. **BUTD-DETR 's contrastive loss helps:** Replacing our contrastive loss with MDETR's results in drop of 2.5% in absolute accuracy.
6. **Concatenating Visual, Language and Object Streams performs worse than a model that has separate streams for each modality** Our motivation is to keep separate streams in 3D cross-modality encoder and decoder to be consistent with 2D BUTD-DETR as explained in Section 2.3.2. We additionally find that having separate streams gives a boost of 0.8%.

Table 2.3: **Results on language grounding in 2D RefCOCO and RefCOCO+ Datasets on Top-1 accuracy metric using standard splits**. All training times are computed using same V100 GPUs. Training epochs are written as $x + y$ where x = number of pre-training epochs and y = number of fine-tuning epochs. All reported results use ResNet101 backbone.

	RefCOCO			RefCOCO+			Training	Training
Method	val	testA	testB	val	testA	testB	Epochs	GPU Hours
UNITER_L [33]	81.4	87.0	74.2	75.9	81.5	66.7	-	-
VILLA_L [87]	82.4	87.5	74.8	76.2	81.5	66.8	-	-
MDETR [158]	**86.8**	**89.6**	81.4	**79.5**	**84.1**	**70.6**	40 + 5	5560
BUTD-DETR (ours)	85.9	88.5	**81.5**	78.2	82.8	70.0	**12 + 5**	**2748**

Table 2.4: **Results on language grounding in Flickr30k 2D images.** We use Recall@k metric. All training times are computed using same V100 GPUs.

	Val			Test			Training	Training
Method	R@1	R@5	R@10	R@1	R@5	R@10	Epochs	GPU hours
VisualBERT [181]	70.4	84.5	86.3	71.3	85.0	86.5	-	-
MDETR [158]	**82.5**	**92.9**	**94.9**	**83.4**	**93.5**	**95.3**	40	5480
BUTD-DETR (ours)	81.2	90.9	92.8	81.0	91.6	93.2	**12**	**2688**

2.4.2 Language grounding in 2D images

We test BUTD-DETR on the referential grounding datasets of RefCOCO [163], RefCOCO+ [388] and Flickr30k entities dataset [265]. We follow the pretrain-then-finetune protocol of MDETR and first pre-train on combined grounding annotations from Flickr30k [265], referring expression datasets [163, 234, 388], Visual Genome [175]. During pre-training the task is to detect all instances of objects mentioned in the utterance. Different than MDETR, we augment this supervision with detection prompts from the MS-COCO dataset [194]. Following MDETR, we directly evaluate our pre-trained model on Flickr30k without any further fine-tuning and fine-tune for 5 epochs on RefCOCO and RefCOCO+.

Table 2.5: **Ablation for BUTD-DETR on the RefCOCO validation set**.

Model	Accuracy
BUTD-DETR	**79.4**
w/o det prompts	77.0
w/o box stream w/o det prompts	76.3
w/o box stream w/o det prompts w/ MDETR's [158] contrastive	74.2

We report top-1 accuracy on the standard splits of RefCOCO and RefCOCO+ in Table 2.3 and Recall metric with ANY-BOX protocol [181] on Flickr30k in Table 2.4. Our model and MDETR use the same 200k image-language pairs from COCO [194], Flickr30k [265] and Visual Genome [175]. VisualBERT [181] is trained on COCO captions. UNITER [33] and VILLA [87] use a larger dataset of 4.4M pairs from COCO, Visual Genome, Conceptual-Captions [308], and SBU Captions [254]. In addition, we augment our training set with detection prompts from COCO. BUTD-DETR trains two times faster than MDETR while getting comparable performance. This computational gain comes mostly from deformable attention which is much cheaper than original visual self-attention that scales quadratically with the number of visual tokens, as already reported in [412].

Ablative analysis

We ablate our model in RefCOCO without pre-training in Table 2.5, since pre-training is computationally expensive due to the size of the combined datasets. Consistent with 3D, removing detection prompts results in an accuracy drop of 2.4%. Additionally removing attention to the box proposal stream results in a drop of 3.1% in accuracy. When replacing our contrastive loss with MDETR's, the model achieves 74.2%, resulting in an additional drop of 2.1% accuracy.

2.4.3 Limitations

Our work relies on language-image alignment and does not address how to ground language better and more robustly through abstraction of the visual features, e.g., the fact that *left* and *right* reverse when we change the user's viewpoint, the fact that numbers requires precise counting, or the fact that the *" chair furthest away from the door"* requires to satisfy a logical constraint which our model can totally violate when presented with out-of-distribution visual input. This limitation is a direct avenue for future work.

2.5 Conclusion

We present BUTD-DETR , a model for referential grounding in 3D and 2D scenes, that attends to language, visual and box proposal streams to decode objects mentioned in the referential utterance and align them to corresponding spans in the input. BUTD-DETR builds upon MDETR [158] and outperforms its straightforward MDETR-3D equivalent by a significant margin thanks to attention on labelled bottom-up box proposals, co-training with detection prompts and improved contrastive losses, setting a new state-of-the-art in two 3D language grounding benchmarks. BUTD-DETR is also the first model in 3D referential grounding that operates on the realistic setup of not having access to oracle object boxes, but rather detects them from the input 3D point cloud.

Chapter 3

ODIN: A Single Model for 2D and 3D Segmentation

In the previous chapter, we saw that BUTD-DETR was able to achieve state of the art results on both 3D and 2D language grounding benchmarks with minimal architectural changes. This result sparks the question: do we really need architectural modifications for a model to operate on 3D and 2D scenes? Are 3D and 2D vision separate fields with different requirements? In this chapter, we go one step further and propose to completely unify the two, with a single model that can segment objects on both 2D images and 3D point clouds.

3.1 Introduction

There has been a surge of interest in porting 2D foundational image features to 3D scene understanding [60, 104, 146, 166, 262, 289, 313, 331, 335]. Some methods lift pre-trained 2D image features using sensed depth to 3D feature clouds [60, 262, 289, 331]. Others distill 2D backbones to differentiable parametric 3D models, e.g., NeRFs, by training them per scene to render 2D feature maps of pre-trained backbones [166, 313]. Despite this effort, and despite the ever-growing power of 2D backbones [36, 370], the state-of-the-art on established 3D segmentation benchmarks such as ScanNet [47]

This chapter is based on the paper previously published at CVPR 2024 [137]

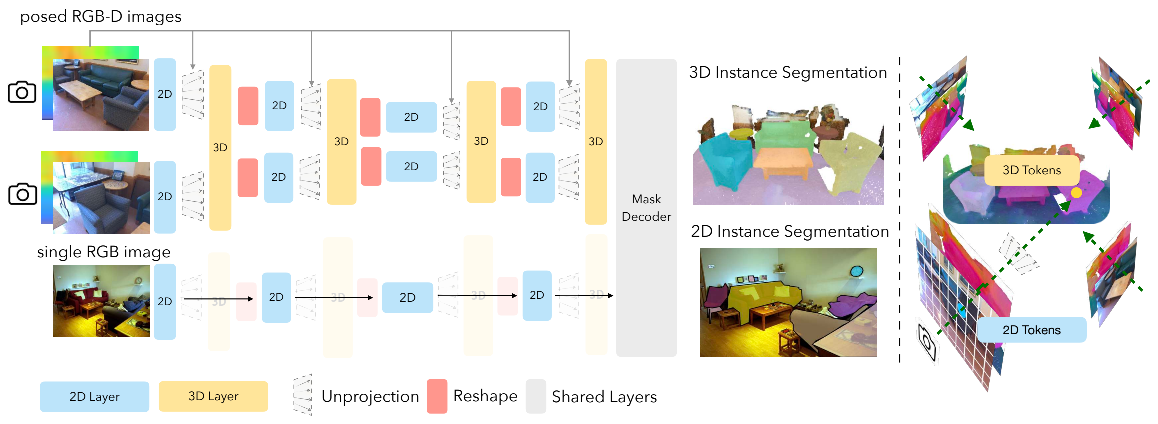


Figure 3.1: **Omni-Dimensional INstance segmentation (ODIN)** is a model that can parse either a single RGB image or a multiview posed RGB-D sequence into 2D or 3D labelled object segments respectively. **Left**: Given a posed RGB-D sequence as input, ODIN alternates between a within-view 2D fusion and a cross-view 3D fusion. When the input is a single RGB image, the 3D fusion layers are skipped. ODIN shares the majority of its parameters across both RGB and RGB-D inputs, enabling the use of pre-trained 2D backbones. **Right**: At each 2D-to-3D transition, ODIN unprojects 2D feature tokens to their 3D locations using sensed depth and camera intrinsics and extrinsics.

and ScanNet200 [294] *still* consists of models that operate directly in 3D, without any 2D pre-training stage [179, 303]. Given the obvious power of 2D pre-training, why is it so difficult to yield improvements in these 3D tasks?

We observe that part of the issue lies in a key implementation detail underlying these 3D benchmark evaluations. Benchmarks like ScanNet do not actually ask methods to use RGB-D images as input, even though this is the sensor data. Instead, these benchmarks first register all RGB-D frames into a single colored point cloud and reconstruct the scene as cleanly as possible, relying on manually tuned stages for bundle adjustment, outlier rejection and meshing, and ask models to label the *output reconstruction*. While it is certainly viable to scan and reconstruct a room before labelling any of the objects inside, this pipeline is perhaps inconsistent with the goals of embodied vision (and typical 2D vision), which involves dealing with actual sensor data and accounting for missing or partial observations. We therefore hypothesize that method rankings will change, and the impact of 2D pre-training will become evident, if we force the 3D models to take posed RGB-D frames as input rather than pre-computed mesh reconstructions. Our revised evaluation setting also opens the door to new methods, which can train and perform inference in either single-view or multi-view settings, with either RGB or RGB-D sensors.

We propose **O**mni-**D**imensional **IN**stance segmentation (ODIN)[†], a model for 2D and 3D object segmentation and labelling that can parse single-view RGB images and/or multiview posed RGB-D images. As shown in fig. 3.1, ODIN alternates between 2D and 3D stages in its architecture, fusing information in 2D within each image view, and in 3D across posed image views. At each 2D-to-3D transition, it unprojects 2D tokens to their 3D locations using the depth maps and camera parameters, and at each 3D-to-2D transition, it projects 3D tokens back to their image locations. Our model differentiates between 2D and 3D features through the positional encodings of the tokens involved, which capture pixel coordinates for 2D patch tokens and 3D coordinates for 3D feature tokens. When dealing with 2D single-view input, our architecture simply skips the 3D layers and makes a forward pass with 2D layers alone.

We test ODIN in 2D and 3D instance segmentation and 3D semantic segmentation on the 2D COCO object segmentation benchmark and the 3D benchmarks of ScanNet [47], ScanNet200 [294], Matterport3D [24], S3DIS [8] and AI2THOR [54, 174]. When compared to methods using pre-computed mesh point cloud as input, our approach performs slightly worse than state-of-the-art on ScanNet and S3DIS, but better on ScanNet200 and Matterport3D. When using real sensor data as input with poses obtained from bundle reconstruction for all methods, our method performs even better, outperforming all prior work by a wide margin, in all datasets. We demonstrate that our model's ability to jointly train on 3D and 2D datasets results in performance increase on 3D benchmarks, and also yields competitive segmentation accuracy on the 2D COCO benchmark. Our ablations show that interleaving 2D and 3D fusion operations outperforms designs where we first process in 2D and then move to 3D, or simply paint 3D points with 2D features. Stepping toward our broader goal of embodied vision, we also deploy ODIN as the 3D object segmentor of a SOTA embodied agent model [301] on the simulation benchmark TEACh [256] in the setup

[†]The Norse god Odin sacrificed one of his eyes for wisdom, trading one mode of perception for a more important one. Our approach sacrifices perception on post-processed meshes for perception on posed RGB-D images.

with access to RGB-D and pose information from the simulator, and demonstrate that our model sets a new state-of-the-art. We make our code publicly available at https://odin-seg.github.io.

3.2 Related Work

3D Instance Segmentation Early methods in 3D instance segmentation [28, 105, 149, 189, 344, 406] group their semantic segmentation outputs into individual instances. Recently, Mask2Former [36] achieved state-of-the-art in 2D instance segmentation by instantiating *object queries*, each directly predicting an instance segmentation mask by doing dot-product with the feature map of the input image. Inspired by it, Mask3D [303] abandons the grouping strategy of prior 3D models to use the simple decoder head of Mask2Former. MAFT [179] and QueryFormer [221] improve over Mask3D by incorporating better query initialization strategies and/or relative positional embeddings. While this shift to Mask2Former-like architecture brought the 3D instance segmentation architectures closer to their 2D counterparts, the inputs and backbones remain very different: 2D models use pre-trained backbones [112, 215], while 3D methods [303] operate over point clouds and use sparse convolution-based backbones [41], trained from scratch on small-scale 3D datasets. In this work, we propose to directly use RGB-D input and design architectures that can leverage strong 2D backbones to achieve strong performance on 3D benchmarks.

3D Datasets and Benchmarks Most 3D models primarily operate on point clouds, avoiding the use of image-based features partly due to the design of popular benchmarks. These benchmarks generate point clouds by processing raw RGB-D sensor data, involving manual and noisy steps that result in misalignments between the reconstructed point cloud and sensor data. For instance, ScanNet [47] undergoes complex mesh reconstruction steps, including bundle reconstruction, implicit TSDF representation fitting, marching cubes, merging and deleting noisy mesh vertices, and finally manual removal of mesh reconstruction with high misalignments. Misalignments introduced by the mesh reconstruction process can cause methods processing sensor data directly to underperform compared to those trained and tested on provided point clouds. Additionally, some datasets, like HM3D [374] lack access to raw RGB-D data. While mesh reconstruction has its applications, many real-time applications need to directly process sensor data.

2D-based 3D segmentation Unlike instance segmentation literature, several approaches for semantic segmentation like MVPNet [144], BPNet [120] and DeepViewAgg [289] utilize the sensor point cloud directly instead of the mesh-sampled point cloud. Virtual Multiview Fusion [177] forgoes sensor RGB-D images in favour of rendering RGB-D images from the provided mesh to fight misalignments and low field-of-view in ScanNet images. Similar to our approach, BPNet and DeepViewAgg integrate 2D-3D information at various feature scales and initialize their 2D streams with pre-trained features. Specifically, they employ separate 2D and 3D U-Nets for processing the respective modalities and fuse features from the two streams through a connection module. Rather than employing distinct streams for featurizing raw data, our architecture instantiates a single unified U-Net which interleaves 2D and 3D layers and can handle both 2D and 3D perception tasks with a single unified architecture. Notably, while these works focus solely on semantic segmentation, our single architecture excels in both semantic and instance segmentation tasks.

Recent advancements in 2D foundation models [172, 276] have spurred efforts to apply them to 3D tasks such as point cloud classification [273, 368, 401], zero-shot 3D semantic segmentation [104, 146, 262] and more recently, zero-shot instance segmentation [331]. Commonly, these methods leverage 2D foundation models to featurize RGB images, project 3D point clouds onto these images, employ occlusion reasoning using depth and integrate features from all views through simple techniques like mean-pooling. Notably, these approaches predominantly focus on semantic segmentation, emphasizing pixel-wise labeling, rather than instance labeling, which necessitates cross-view reasoning to associate the same object instance across multiple views. OpenMask3D [331] is the only method that we are aware of that attempts 3D instance segmentation using 2D foundation models, by training a class-agnostic 3D object segmentor on 3D point clouds and labelling it utilizing CLIP features. Despite their effectiveness in a zero-shot setting, they generally lag behind SOTA 3D supervised methods by 15-20%. Rather than relying on features from foundation models, certain works [64, 90] create 3D pseudo-labels using pre-trained 2D models. Another line of work involves fitting Neural-Radiance Fields (NeRFs), incorporating features from CLIP [166, 335] or per-view instance segmentations from state-of-the-art

2D segmentors [313]. These approaches require expensive per-scene optimization that prohibits testing on all test scenes to compare against SOTA 3D discriminative models. Instead of repurposing 2D foundation models for 3D tasks, Omnivore [92] proposes to build a unified architecture that can handle multiple visual modalities like images, videos and single-view RGB-D image but they only show results for classification tasks. We similarly propose a single unified model capable of performing both single-view 2D and multi-view 3D instance and semantic segmentation tasks while utilizing pre-trained weights for the majority of our architecture.

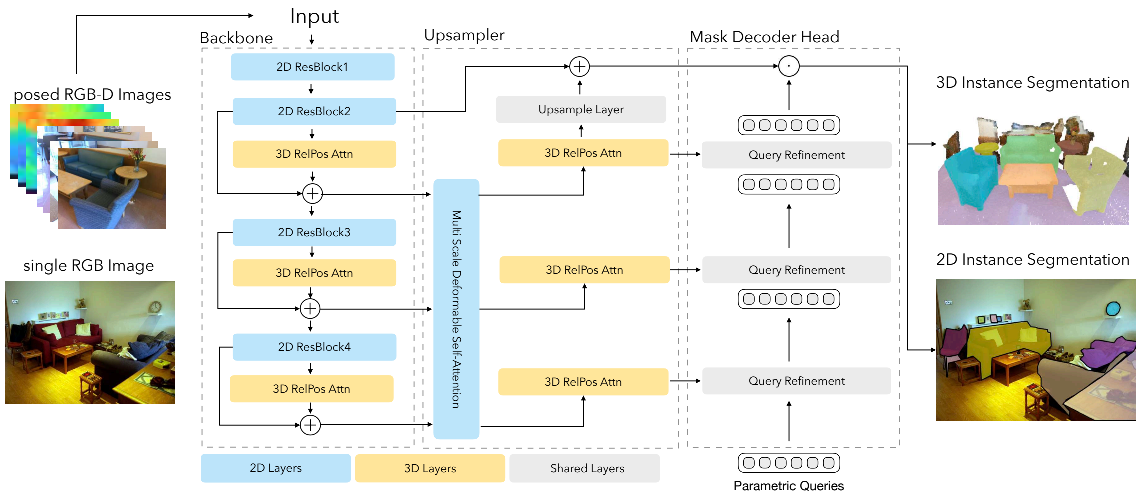


Figure 3.2: **ODIN Architecture**: The input to our model is either a single RGB image or a multiview RGB-D posed sequence. We feed them to ODIN's backbone which interleaves 2D within-view fusion layers and 3D cross-view attention layers to extract feature maps of different resolutions (scales). These feature maps exchange information through a multi-scale attention operation. Additional 3D fusion layers are used to improve multiview consistency. Then, a mask decoder head is used to initialize and refine learnable slots that attend to the multi-scale feature maps and predict object segments (masks and semantic classes).

3.3 Method

ODIN's architecture is shown in fig. 3.2. It takes either a single RGB image or a set of posed RGB-D images (i.e., RGB images associated with depth maps and camera parameters) and outputs the corresponding 2D or 3D instance segmentation masks and their semantic labels. To achieve this, ODIN alternates between a 2D within-view fusion and a 3D attention-based cross-view fusion, as illustrated in blue blocks and yellow blocks in fig. 3.2. A segmentation decoding head predicts instance masks and semantic labels. Notably, ODIN shares the majority of its parameters across both RGB and multiview RGB-D inputs. We detail the components of our architecture below.

Within-view 2D fusion: We start from a 2D backbone, such as ResNet50 [112] or Swin Transformer [215], pre-trained for 2D COCO instance segmentation following Mask2Former [36], a state-of-the-art 2D segmentation model. When only a single RGB image is available, we pass it through the full backbone to obtain 2D features at multiple scales. When a posed RGB-D sequence is available, this 2D processing is interleaved with 3D stages, described next. By interleaving within-view and cross-view contextualization, we are able to utilize the pre-trained features from the 2D backbone while also fusing features across views, making them 3D-consistent.

Cross-view 3D fusion: The goal of cross-view fusion is to make the individual images' representations consistent across views. As we show in our ablations, cross-view feature consistency is essential for 3D instance segmentation: it enables the segmentation head to realize that a 3D object observed from multiple views is indeed a single instance, rather than a separate instance in each viewpoint.

1. 2D-to-3D Unprojection: We unproject each 2D feature map to 3D by lifting each feature vector to a corresponding 3D location, using nearest neighbor depth and known camera intrinsic and extrinsic parameters, using a pinhole camera model. Subsequently, the resulting featurized point cloud undergoes voxelization, where the 3D space is discretized into a volumetric grid. Within each occupied grid cell (voxel), the features and XYZ coordinates are mean-pooled to derive new sets of 3D feature tokens and their respective 3D locations.

2. 3D k-NN Transformer with Relative Positions: We fuse information across 3D tokens using k-nearest-neighbor attention with relative 3D positional embeddings. This is similar to Point Transformers [360, 403], but we simply use vanilla cross-attention instead of the vector attention proposed in those works. Specifically, in our approach, each 3D token attends to its k nearest neighbors. The positional embeddings in this operation are relative to the query token's location. We achieve this by encoding the distance vector between a token and its neighbour with an

MLP. The positional embedding for the query is simply encoding of the 0 vector. We therefore have

$$\text{query}_{\text{pos}} = \text{MLP}(0); \tag{3.1}$$

$$\text{key}_{\text{pos}} = \text{MLP}(p_i - p_j), \tag{3.2}$$

where p_i represents the 3D tokens, shaped $N \times 1 \times 3$, and p_j represents the k nearest neighbors of each p_i, shaped $N \times k \times 3$. In this way, the attention operation is invariant to the absolute coordinates of the 3D tokens and only depends on their relative spatial arrangements. While each 3D token always attends to the same k neighbors, its effective receptive field grows across layers, as the neighbors' features get updated when they perform their own attention [72].

3. 3D-to-2D Projection: After contextualizing the tokens in 3D, we project the features back to their original 2D locations. We first copy the feature of each voxel to all points within that voxel. We then reshape these points back into multiview 2D feature maps, so that they may be processed by the next 2D module. The features vectors are unchanged in this transition; the difference lies in their interpretation and shape. In 2D the features are shaped $V \times H \times W \times F$, representing a feature map for each viewpoint, and in 3D they are shaped $N \times F$, representing a unified feature cloud, where $N = V \cdot H \cdot W$.

Cross-scale fusion and upsampling: After multiple single-view and cross-view stages, we have access to multiple features maps per image, at different resolutions. We merge these with the help of deformable 2D attention, akin to Mask2Former [36], operating on the three lowest-resolution scales $(1/32, 1/16, 1/8)$. When we have 3D input, we apply an additional 3D fusion layer at each scale after the deformable attention, to restore the 3D consistency. Finally, we use a simple upsampling layer on the 1/8 resolution feature map to bring it to 1/4 resolution and add with a skip connection to the 1/4 feature map from the backbone.

Sensor depth to mesh point cloud feature transfer: For 3D benchmarks like ScanNet [47] and ScanNet200 [294], the objective is to label a point cloud derived from a mesh rather than the depth map from the sensor. Hence, on those benchmarks, instead of upsampling the 1/8 resolution feature map to 1/4, we trilinearly interpolate features from the 1/8 resolution feature map to the provided point cloud sampled from the mesh. This means: for each vertex in the mesh, we trilinearly interpolate from our computed 3D features to obtain interpolated features. We additionally similarly interpolate from the unprojected 1/4 resolution feature map in the backbone, for an additive skip connection.

Shared 2D-3D segmentation mask decoder: Our segmentation decoder is a Transformer, similar to Mask2Former's decoder head, which takes as input upsampled 2D or 3D feature maps and outputs corresponding 2D or 3D segmentation masks and their semantic classes. Specifically, we instantiate a set of N learnable object queries responsible for decoding individual instances. These queries are iteratively refined by a *Query Refinement* block, which consists of cross-attention to the upsampled features, followed by a self-attention between the queries. Except for the positional embeddings, all attention and query weights are shared between 2D and 3D. We use Fourier positional encodings in 2D, while in 3D we encode the XYZ coordinates of the 3D tokens with an MLP. The refined queries are used to predict instance masks and semantic classes. For mask prediction, the queries do a token-wise dot product with the highest-resolution upsampled features. For semantic class prediction, we use an MLP over the queries, mapping them to class logits. We refer readers to Mask2Former [36] for further details.

Open vocabulary class decoder: Drawing inspiration from prior open-vocabulary detection methods [134, 182, 415], we introduce an alternative classification head capable of handling an arbitrary number of semantic classes. This modification is essential for joint training on multiple datasets. Similar to BUTD-DETR [134] and GLIP [182], we supply the model with a *detection prompt* formed by concatenating object categories into a sentence (e.g., *"Chair. Table. Sofa."*) and encode it using RoBERTa [213]. In the query-refinement block, queries additionally attend to these text tokens before attending to the upsampled feature maps. For semantic class prediction, we first perform a dot-product operation between queries and language tokens, generating one logit per token in the detection prompt. The logits corresponding to prompt tokens for a specific object class are then averaged to derive per-class logits. This can handle multi-word noun phrases such as *"shower curtain"*, where we average the logits corresponding to *"shower"* and *"curtain"*. The segmentation masks are predicted by a pixel-/point-wise dot-product, in the same fashion as described earlier.

Point Cloud	Model	mAP	mAP50	mAP25
Sensor RGBD	Mask3D[§] [303]	43.9	60.0	69.9
Sensor RGBD	ODIN-ResNet50 (Ours)	47.8	69.8	**83.6**
Sensor RGBD	ODIN-Swin-B (Ours)	**50.0**	**71.0**	**83.6**
Mesh Sampled	SoftGroup [344]	46.0	67.6	78.9
Mesh Sampled	PBNet [406]	54.3	70.5	78.9
Mesh Sampled	Mask3D [303]	55.2	73.7	**83.5**
Mesh Sampled	QueryFormer [221]	56.5	74.2	83.3
Mesh Sampled	MAFT [179]	**58.4**	**75.9**	-

Table 3.1: **ScanNet Instance Segmentation Task.** ([§] = trained by us using official codebase)

Implementation details: We initialize our model with pre-trained weights from Mask2Former [36] trained on COCO [192]. Subsequently, we train all parameters end-to-end, including both pre-trained and new parameters from 3D fusion layers. During training in 3D scenes, our model processes a sequence of N consecutive frames, usually comprising 25 frames. At test time, we input all images in the scene to our model, with an average of 90 images per scene in ScanNet. We use vanilla closed-vocabulary decoding head for all experiments except when training jointly on 2D-3D datasets. There we use our open vocabulary class decoder that lets us handle different label spaces in these datasets. During training, we employ open vocabulary mask decoding for joint 2D and 3D datasets and vanilla closed-vocabulary decoding otherwise. Training continues until convergence on 2 NVIDIA A100s with 40 GB VRAM, with an effective batch size of 6 in 3D and 16 in 2D. For joint training on 2D and 3D datasets, we alternate sampling 2D and 3D batches with batch sizes of 3 and 8 per GPU, respectively. We adopt Mask2Former's strategy, using Hungarian matching for matching queries to ground truth instances and supervision losses. While our model is only trained for instance segmentation, it can perform semantic segmentation for free at test time like Mask2Former. We refer to Mask2Former [36] for more details.

Point Cloud	Model	mIoU
Sensor RGBD	MVPNet [144]	68.3
Sensor RGBD	BPNet [120]	69.7
Sensor RGBD	DeepViewAgg [289]	71.0
Sensor RGBD	ODIN-ResNet50 (Ours)	73.3
Sensor RGBD	ODIN-Swin-B (Ours)	**77.8**
Rendered RGBD	VMVF [177]	**76.4**
Mesh Sampled	Point Transformer v2 [360]	75.4
Mesh Sampled	Stratified Transformer [178]	74.3
Mesh Sampled	OctFormer [348]	75.7
Mesh Sampled	Swin3D-L [377]	**76.7**
Mesh Sampled (Zero-Shot)	OpenScene [262]	54.2

Table 3.2: **ScanNet Semantic Segmentation Task.**

Point Cloud	Model	mAP	mAP50	mAP25
Sensor RGBD	Mask3D [303] §	15.5	21.4	24.3
Sensor RGBD	ODIN-ResNet50 (Ours)	25.6	36.9	43.8
Sensor RGBD	ODIN-Swin-B (Ours)	**31.5**	**45.3**	**53.1**
Mesh Sampled	Mask3D [303]	27.4	37.0	42.3
Mesh Sampled	QueryFormer [221]	28.1	37.1	**43.4**
Mesh Sampled	MAFT [179]	**29.2**	**38.2**	43.3
Mesh Sampled (Zero-Shot)	OpenMask3D [331]	15.4	19.9	23.1

Table 3.3: **ScanNet200 Instance Segmentation Task.** (§ = trained by us using official codebase)

3.4 Experiments

3.4.1 Evaluation on 3D benchmarks

Datasets: First, we test our model on 3D instance and semantic segmentation in the ScanNet [47] and ScanNet200 [294] benchmarks. The goal of these benchmarks is to label the point cloud extracted from the 3D mesh of a scene reconstructed from raw sensor data. ScanNet evaluates on 20 common semantic classes, while ScanNet200 uses 200 classes, which is more representative of the long-tailed object distribution encountered in the real world. We report results on the official validation split of these datasets here and on the official test split in the supplementary.

Evaluation metrics: We follow the standard evaluation metrics, namely mean Average Precision (mAP) for instance segmentation and mean Intersection over Union

Point Cloud	Model	mIoU
Sensor RGBD	ODIN-ResNet50 (Ours)	35.8
Sensor RGBD	ODIN-Swin-B (Ours)	**40.5**
Mesh Sampled	LGround [294]	28.9
Mesh Sampled	CeCo [408]	32.0
Mesh Sampled	Octformer [348]	**32.6**

Table 3.4: **ScanNet200 Semantic Segmentation Task.**

(mIoU) for semantic segmentation.

Baselines: In *instance segmentation*, our main baseline is the SOTA 3D method Mask3D [303]. For a thorough comparison, we train both Mask3D and our model with sensor RGB-D point cloud input and evaluate them on the benchmark-provided mesh-sampled point clouds. We also compare with the following recent and concurrent works: PBNet [406], QueryFormer [221] and MAFT [179]. QueryFormer and MAFT explore query initialization and refinement in a Mask3D-like architecture and thus have complementary advantages to ours. Unlike ODIN, these methods directly process 3D point clouds and initialize their weights from scratch. As motivated before, utilizing RGB-D input directly has several advantages, including avoiding costly mesh building processes, achieving closer integration of 2D and 3D perception, and leveraging pre-trained features and abundant 2D data.

In *semantic segmentation*, we compare with MVPNet [144], BPNet [120] and state-of-the-art DeepViewAgg [289] which directly operate on sensor RGB or RGB-D images and point clouds. We also compare with VMVF [177] which operates over rendered RGB-D images from the provided mesh, with heuristics for camera view sampling to avoid occlusions, ensures balanced scene coverage, and employs a wider field-of-view, though we note their code is not publicly available. Similar to ODIN, all of these methods utilize 2D pre-trained backbones. We also compare with Point-Transformer v2 [360], Stratified Transformer [178], OctFormer [348] and Swin3D-L [377] which process the mesh-sampled point cloud directly, without using any 2D pre-training. On the ScanNet200 semantic segmentation benchmark, we compare with SOTA OctFormer [348] and with CeCo [408], a method specially designed to fight class-imbalance in ScanNet200. These methods directly process the point cloud and do not use 2D image pre-trained weights. We also compare with LGround [294] which uses 2D CLIP pre-training. We also compare with zero-shot 2D foundation model-based 3D models of OpenScene [262] and OpenMask3D [331]. This comparison is unfair since they are not supervised within-domain, but we include them for completeness. We draw the following conclusions:

Performance drops with sensor point cloud as input (table 3.1): Mask3D's performance drops from 55.2% mAP with mesh point cloud input to 43.9% mAP with sensor point cloud input. This is consistent with prior works [177, 289] in 3D semantic segmentation on ScanNet, which attributes the drop to misalignments caused by noise in camera poses, depth variations and post-processing steps.

ODIN outperforms SOTA 3D methods with sensor point cloud input and underperforms them when baselines use mesh-sampled point clouds (table 3.1): Our model significantly outperforms SOTA Mask3D model with sensor point cloud input and achieves comparable performance to methods using mesh-sampled point cloud input on the mAP25 metric while far behind on mAP metric, due to misalignments between the 3D mesh and the sensor point cloud.

ODIN sets a new SOTA in semantic segmentation on ScanNet (table 3.2) outperforming all methods on all setups including the models trained on the sensor, rendered and mesh sampled point clouds.

ODIN sets a new instance segmentation SOTA on the long-tailed Scan-Net200 dataset (table 3.3) outperforming SOTA 3D models on all setups including the models trained on mesh-sampled point cloud, especially by a large margin in mAP25 metric, while exclusively utilizing sensor RGB-D data. This highlights the contribution of 2D features, particularly in detecting a long tail of class distribution where limited 3D data is available. We show more detailed results with performance on the head, common and tail classes in the appendix (section B.1.3).

ODIN sets a new semantic segmentation SOTA on ScanNet200 (table 3.4), outperforming SOTA semantic segmentation models that use mesh point clouds.

3.4.2 Evaluation on multiview RGB-D in simulation

Using the AI2THOR [174] simulation environment with procedural homes from ProcThor [54], we collected RGB-D data for 1500 scenes (1200 training, 300 test)

Model	mAP	mAP50	mAP25	mIoU
Mask3D [303]	60.6	70.8	76.6	-
ODIN-ResNet50 (Ours)	63.8	**73.8**	**80.2**	**71.5**
ODIN-Swin-B (Ours)	**64.3**	73.7	78.6	71.4

Table 3.5: **AI2THOR Semantic and Instance Segmentation.**

	TEACh				**ALFRED**			
	Unseen		**Seen**		**Unseen**		**Seen**	
	SR	GC	SR	GC	SR	GC	SR	GC
FILM [239]	-	-	-	-	30.7	42.9	26.6	38.2
HELPER [301]	15.8	14.5	11.6	19.4	37.4	55.0	26.8	41.2
HELPER + ODIN (OURS)	**18.6**	**18.6**	**13.8**	**26.6**	**47.7**	**61.6**	**33.5**	**47.1**

Table 3.6: **Embodied Instruction Following.** SR = success rate. GC = goal condition success rate.

of similar size as ScanNet (more details in appendix, section B.2). We train and evaluate our model and SOTA Mask3D [303] on the unprojected RGB-D images. As shown in table 3.5, our model outperforms Mask3D by 3.7% mAP, showing strong performance in a directly comparable RGB-D setup. It suggests that current real-world benchmarks may restrain models that featurizes RGB-D sensor point clouds due to misalignments. We hope this encourages the community to also focus on directly collecting, labeling, and benchmarking RGB-D sensor data.

3.4.3 Embodied Instruction Following

We apply ODIN in the embodied setups of TEACh [256] and ALFRED [310] where agents have access to RGB, depth and camera poses and need to infer and execute task and action plans from dialogue segments and instructions, respectively. These agents operate in dynamic home environments and cannot afford expensive mesh building steps. Detecting objects well is critical for task success in both cases. Prior SOTA methods [256, 301] run per-view 2D instance segmentation models [36, 62] and link the detected instances using simple temporal reasoning regarding spatial and appearance proximity. Instead, ODIN processes its last N egocentric views and

	ScanNet			COCO
	mAP	mAP50	mAP25	mAP
Mask3D [303]	43.9	60.0	69.9	✗
Mask2Former [36]	✗	✗	✗	**43.7**
ODIN (trained in 2D)	✗	✗	✗	43.6
ODIN (trained in 3D)	47.8	69.8	**83.6**	✗
ODIN (trained jointly)	**49.1**	**70.1**	83.1	41.2

Table 3.7: **Joint Training on Sensor RGB-D point cloud from ScanNet and 2D RGB images from COCO.**

Model	mAP	mIoU
ODIN (Ours)	**47.8**	73.3
No 3D Fusion	39.3	73.2
No interleaving	41.7	**73.6**

Table 3.8: **Cross-View Contextualization** on ScanNet Dataset.

segments objects instances directly in 3D. We equip HELPER [301], a state-of-the-art embodied model, with ODIN as its 3D object detection engine. We evaluate using Task Sucess Rate (SR) which checks if the entire task is executed successfully, and Goal Conditioned Success Rate (GC) which checks the proportion of satisfied subgoals across all episodes [256, 310]. We perform evaluation on ”valid-seen” (houses similar to the training set) and ”valid-unseen” (dissimilar houses) splits. In table 3.6, we observe that HELPER with ODIN as its 3D object detector significantly outperforms HELPER that uses the original 2D detection plus linking perception pipeline.

3.4.4 Ablations and Variants

We conduct our ablation experiments on the ScanNet dataset in table 3.7 and tables 3.8, 3.9, 3.10. Our conclusions are:

Joint 2D-3D training helps 3D perception We compare joint training of ODIN on sensor RGB-D point clouds from ScanNet and 2D RGB images from COCO to variants trained independently on 2D and 3D data, all initialized from pre-trained COCO weights. Since there are different classes in ScanNet and COCO, we use our open-vocabulary semantic class-decoding head instead of the vanilla closed-vocabulary head.

Model	mAP	mIoU
ODIN (Ours)	**47.8**	**73.3**
Only pre-trained backbone	42.3	72.9
No pre-trained features	41.5	68.6

Table 3.9: **Effect of Pre-Trained Features** on ScanNet Dataset.

Model	ResNet50		Swin-B	
	mAP	mIoU	mAP	mIoU
ODIN (Ours)	**47.8**	73.3	**50.0**	**77.8**
With frozen backbone	46.7	**74.3**	46.2	75.9

Table 3.10: **Effect of Freezing Backbone** on ScanNet Dataset.

Results in table 3.7 show that joint training yields a 1.3% absolute improvement in 3D, and causes a similar drop in 2D. This experiment indicates that a single architecture can perform well on both 2D and 3D tasks, thus indicating that we may not need to design vastly different architectures in either domain. However, the drop in 2D performance indicates a potential for further improvements in the architecture design to retain the performance in the 2D domain. Nevertheless, this experiment highlights the benefits of joint training with 2D datasets for 3D segmentation in ODIN. Note that we do not jointly train on 2D and 3D datasets in any of our other experiments due to computational constraints.

Cross-View fusion is crucial for instance segmentation but not for semantic segmentation (table 3.8): removing 3D cross-view fusion layers results in an 8.5% mAP drop for instance segmentation, without any significant effect in semantic segmentation. Popular 2D-based 3D open vocabulary works [146, 262] without strong cross-view fusion only focus on semantic segmentation and thus could not uncover this issue. Row-3 shows a 6.1% mAP drop when cross-view 3D fusion happens after all within-view 2D layers instead of interleaving the within-view and cross-view fusion.

2D pre-trained weight initialization helps (table 3.9): initializing only the image backbone with pre-trained weights, instead of all layers (except the 3D fusion layers), results in a 5.5% mAP drop (row-2). Starting the entire model from scratch leads to a larger drop of 6.3% mAP (row-3). This underscores the importance of sharing as many parameters as possible with the 2D models to leverage the maximum possible

2D pre-trained weights.

Stronger 2D backbones helps (table 3.10): using Swin-B over ResNet50 leads to significant performance gains, suggesting that ODIN can directly benefit from advancements in 2D computer vision.

Finetuning everything including the pre-trained parameters helps (table 3.10): ResNet50's and Swin's performance increases substantially when we fine-tune all parameters. Intuitively, unfreezing the backbone allows 2D layers to adapt to cross-view fused features better. Thus, we keep our backbone unfrozen in all experiments.

Supplying 2D features directly to 3D models does not help: Concatenating 2D features with XYZ+RGB as input to Mask3D yields 53.8% mAP performance, comparable to 53.3% of the baseline model with only XYZ+RGB as input.

3.4.5 Additional Experiments

We show evaluations on the hidden test set of ScanNet and ScanNet200 in section B.1.1, results and comparisons with baselines on S3DIS [8] and MatterPort3D [24] datasets in section B.1.2 and performance gains in 2D perception with increasing context views in section B.1.4.

3.4.6 Limitations

Our experiments reveal the following limitations for ODIN: Firstly, like other top-performing 3D models, it depends on accurate depth and camera poses. Inaccurate depth or camera poses cause a sharp decrease in performance (similar to other 3D models, like Mask3D). In our future work, we aim to explore unifying depth and camera pose estimation with semantic scene parsing, thus making 3D models more resilient to noise. Secondly, in this work, we limited our scope to exploring the design of a unified architecture without scaling up 3D learning by training on diverse 2D and 3D datasets jointly. We aim to explore this in future to achieve strong generalization to in-the-wild scenarios, akin to the current foundational 2D perception systems. Our results suggest a competition between 2D and 3D segmentation performance

†We do not use the expensive DB-SCAN post-processing of Mask3D, and hence it gets 53.3% mAP instead of 55.2% as reported by their paper when training ODIN jointly on both modalities. Exploring ways to make 2D and 3D training more synergistic is a promising avenue for future work.

3.5 Conclusion

We presented ODIN, a model for 2D and 3D instance segmentation that can parse 2D images and 3D point clouds alike. ODIN represents both 2D images and 3D feature clouds as a set of tokens that differ in their positional encodings which represent 2D pixel coordinates for 2D tokens and 3D XYZ coordinates for 3D tokens. Our model alternates between within-image featurization and cross-view featurization. It achieves SOTA performance in ScanNet200 and AI2THOR instance segmentation benchmarks, outperforms all methods operating on sensor point clouds and achieves competent performance to methods operating over mesh-sampled pointcloud. Our experiments show that ODIN outperforms alternative models that simply augment 3D point cloud models with 2D image features as well as ablative versions of our model that do not alternate between 2D and 3D information fusion, do not co-train across 2D and 3D and do no pre-train the 2D backbone.

Chapter 4

Analogy-Forming Transformers for Few-Shot 3D Parsing

Ask not what it is, ask what it is like.

Moshe Bar

While ODIN focused on unifying 3D and 2D architectures, here we take an orthogonal step; building again on top of successful Detection Transformer architectures, we introduce memory prompting and conditioning for instance segmentation in 3D point clouds. This enables fast in-context learners that can uncover 3D correspondences without even being trained to do so.

4.1 Introduction

The dominant paradigm in existing deep visual learning is to train high-capacity networks that map input observations to task-specific outputs. Despite their success across a plethora of tasks, these models struggle to perform well in *few-shot* settings where only a small set of examples are available for learning. Meta-learning approaches provide one promising solution to this by enabling efficient task-specific adaptation of generic models, but this specialization comes at the cost of poor performance on

This chapter is based on the paper previously published at ICLR 2023 [94] the original tasks as well as the need to adapt separate models for each novel task.

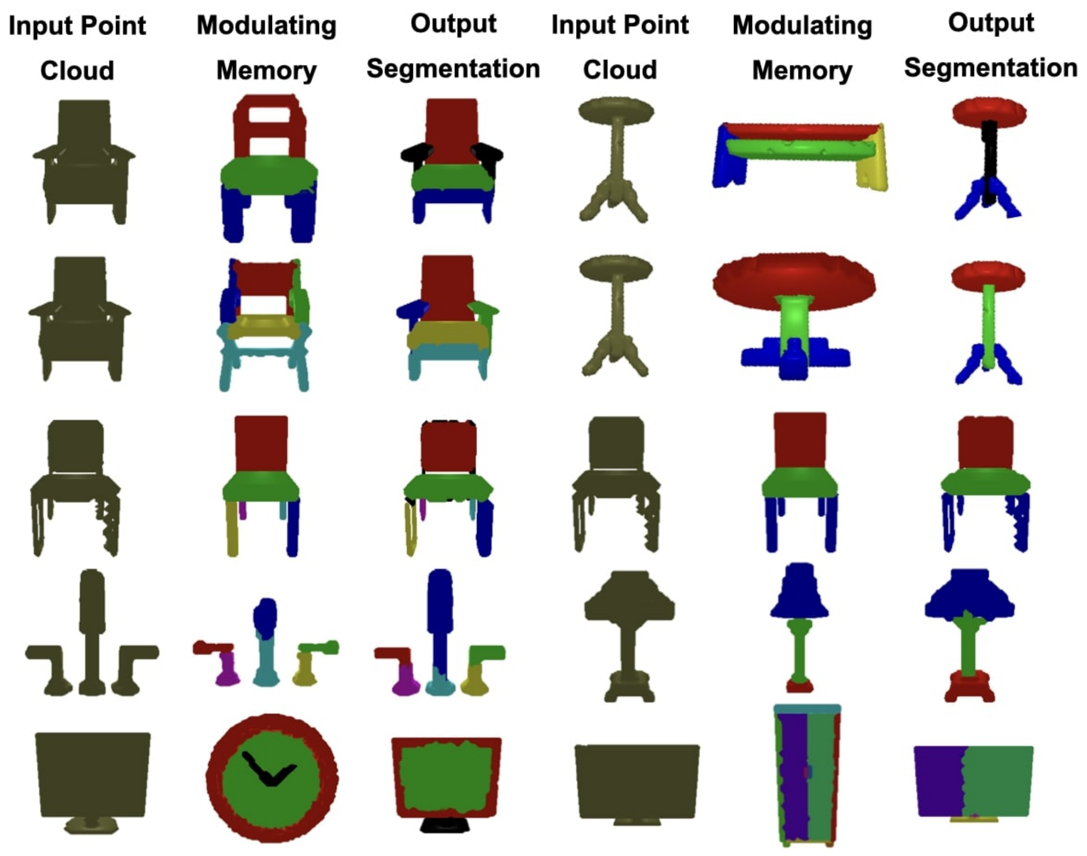


Figure 4.1: **Analogical Networks form analogies between retrieved memories and the input scene** by using memory part encodings as queries to localize corresponding parts in the scene. Retrieved memories (2nd and 5th columns) modulate segmentation of the input 3D point cloud (1st and 4th columns, respectively). We indicate corresponding parts between the memory and the input scene with the same color. Cross-object part correspondences emerge even without any part association or semantic part labelling supervision (5th row). For example, the model learns to correspond the parts of a clock and a TV set, without ever trained with such cross scene part correspondence. Parts shown in black in columns 3 and 6 are decoded from scene-agnostic queries and thus they are not in correspondence to any parts of the memory scene. Conditioning the same input point cloud on memories with finer or coarser labelling results in segmentation of analogous granularity (3rd row).

We introduce Analogical Networks, a semi-parametric learning framework for 3D scene parsing that pursues analogy-driven prediction: instead of mapping the input scene to part segments directly, the model reasons analogically and maps the input to modifications and compositions of past labelled visual experiences. Analogical Networks encode domain knowledge explicitly in a collection of structured labelled scene memories as well as implicitly, in model parameters. Given an input 3D scene, the model retrieves relevant memories and uses them to modulate inference and segment object parts in the input point cloud. During modulation, the input scene and the retrieved memories are jointly encoded and contextualized via cross-attention operations. The contextualized memory part features are then used to segment analogous parts in the 3D input scene, binding the predicted part structure to the one from memory, as shown in Figure 4.1. Given the same input scene, the output prediction changes with varying conditioning memories. For example, conditioned on visual memories of varying label granularity, the model segments the input in a granularity analogous to the one of the retrieved memory. One-shot, few-shot or many-shot learning are treated uniformly in Analogical Networks, by conditioning on the appropriate set of memories. This is very beneficial over methods that specifically target few-shot only scenarios, since, at test time, an agent usually cannot know whether a scene is an example of many-shot or few-shot categories.

Analogical Networks learn to bind memory part features to input scene part segments. Fine-grained part correspondence annotations across two 3D scenes are not easily available. We devise a novel within-scene pre-training scheme to encourage correspondence learning across scenes. We augment (rotate and deform) a given scene in two distinct ways, and train the modulator to parse one of them given the other as its modulating memory, bypassing the retrieval process. During this within-scene training, we have access to the part correspondence between the memory and the input scene, and we use it to supervise the query-to-part assignment process. We show within-scene training helps our model learn to associate memory queries to similarly labelled parts *across scenes* without ever using cross-scene part correspondence annotations, as shown in Figure 4.1.

We test our model on the PartNet benchmark of Mo et al. [243] for 3D object segmentation. We compare against state-of-the-art (SOTA) 3D object segmentors, as well as meta-learning and few-shot learning [319] baselines adapted for the task of 3D parsing. Our experiments show that Analogical Networks perform similarly to the parametric alone baselines in the standard many-shot train-test split and particularly shine over the parametric baselines in the few-shot setting: Analogical Networks segment novel instances much better than parametric existing models, simply by expanding the memory repository with encodings of a few exemplars, even without any weight updates. We further compare against variants of our model that consider memory retrieval and attention without memory query binding, and thus lack explicit analogy formation, as well as other ablative versions of our model to quantify the contribution of the retriever and the proposed within-scene memory-augmented pre-training. Our code and models are publicly available in the project webpage: http://analogicalnets.github.io/.

4.2 Related work

Few-shot prediction: meta-learning and learning associations A key goal for our approach is to enable accurate inference in few-shot settings. Previous approaches [11, 78, 214, 230, 251, 295, 319, 333, 352, 353, 375] that target similar settings can be broadly categorized as relying on either meta-learning or learning better associations. Meta-learning approaches tackle few-shot prediction by learning a generic model that can be efficiently adapted to a new task of interest from a few labelled examples. While broadly applicable, these methods result in catastrophic forgetting of the original task during adaptation and thus require training a new model for each task of interest. Moreover, the goal of learning generic and rapidly adaptable models can lead to suboptimal performance over the base tasks with abundant data. An alternative approach for the few-shot setting is to learn better associations. For example, the category of a new example may be inferred by transferring the label(s) from the one (or few) closest samples [319, 329]. While this approach obviates the need for adapting models and can allow prediction in few-shot and many-shot settings, the current approaches are only applicable to global prediction, e.g., image labels. Our work can be viewed as extending such association-based methods to allow predicting fine-grained and generic visual structures using our proposed modulation-based prediction mechanism.

Memory-augmented neural networks Analogical Networks is a type of memory-augmented neural networks. Memory-augmentation of parametric models permits **fast learning** with few examples, where data are saved in and can be accessed from the memory immediately after their acquisition [300], while learning via parameter update is slow and requires multiple gradient iterations on de-correlated examples. External memories have recently been used to scale up language models [17, 167], and alleviate from the limited context window of parametric transformers [361], as well as to store knowledge in the form of entity mentions [53], knowledge graphs [52] and question-answer pairs [30]. Memory attention layers in these models are used to influence the computation of transformer layers and have proven very successful for factual question answering, but also for sentence completion over their parametric counterparts.

In-context learning In-context learning (ICL) [22] aims to induce a model to perform a task by feeding in input-output examples along with an unlabeled query example. The primary advantage of in-context learning is that it enables a single model to perform many tasks immediately without fine-tuning. Analogical Networks are in-context learners in that they infer the part segmentation of an object 3D point cloud in the context of retrieved labelled object 3D point clouds. While in language models ICL emerges at test time while training unsupervised (and out-of-context) for language completion, Analogical Networks are trained in-context with related examples using supervision and self-supervision. Although in prompted language models [22, 127] the input-output pairs are currently primarily decided by the engineer, in Analogical Networks they are automatically inferred by the retriever.

4.3 Analogical Networks for 3D object parsing

The architecture of Analogical Networks is illustrated in Figure 4.2. Analogical Networks are analogy-forming transformer networks for part segmentation where queries that decode entities in the input scene are supplied by retrieved part-encoded labelled scenes, as well as by the standard set of scene-agnostic parametric queries of detection transformers [23]. When a memory part query is used to decode a part segmentation in the input scene, we say the two parts, in the memory and the input, are put into correspondence. By using memory queries to decode parts in the input, our model forms analogies between detected part graphs in the input scene and part graphs in the modulating memories. Then, metadata, such as semantic labels, attached on memory queries automatically propagate to the detected parts.

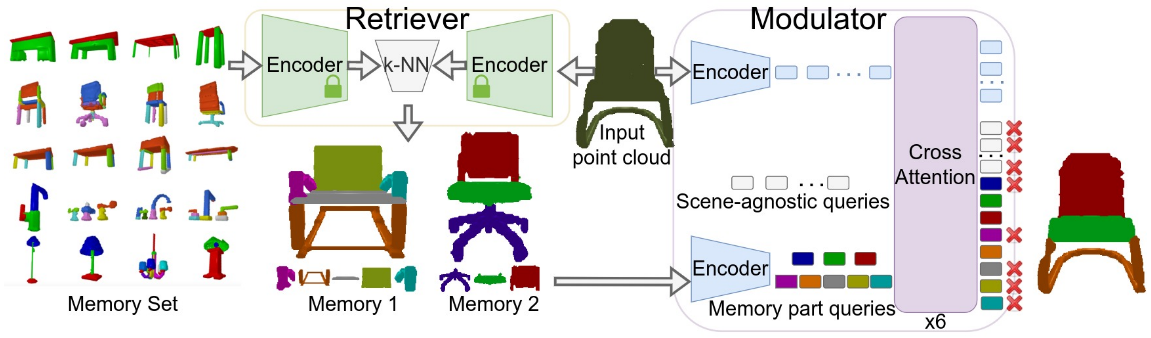


Figure 4.2: **Architecture for Analogical Networks.** Analogical Networks are comprised of retriever and modulator sub-networks. In the retriever, labelled memories and the (unlabelled) input point cloud are separately encoded into feature embeddings and the top-k most similar memories to the present input are retrieved. In the modulator, each retrieved memory is encoded into a set of part feature embeddings and initializes a query that is akin to a slot to be "filled" with the analogous part entity in the present scene. These queries are appended to a set of learnable parametric scene-agnostic queries. The modulator contextualizes the queries with the input point cloud through iterative self and cross-attention operations that also update the point features of the input. When a memory part query decodes a part in the input point cloud, we say the two parts are put into correspondence by the model. We color them with the same color to visually indicate this correspondence.

Analogical Networks are comprised of two main modules: (i) A retriever, that takes as input a 3D object point cloud and a memory repository of labelled 3D object point clouds, and outputs a set of relevant memories for the scene at hand, and (ii) a modulator, that jointly encodes the memories and the input scene and predicts its 3D part segmentation.

Retriever The retriever has access to a memory repository of labelled 3D object point clouds. Each labelled training example is a memory in this repository. Examples labelled with different label granularities constitute separate memories. The retriever encodes each memory example as well as the input point cloud into distinct normalized 1D feature encodings. The top-k memories are retrieved by computing an inner product between the input point cloud feature and the memory features.

Modulator The modulator takes as input the retrieved memories and the unlabelled input point cloud and predicts part segments. The input scene is encoded into a set of 3D point features. Each memory scene is encoded into a set of 1D part encodings, one for each annotated part in the memory, which we call memory part queries, in accordance to parametric queries used to decode objects in DETR [23]. We additionally use scene-agnostic parametric queries, which are learnable slots that decode parts the memory cannot explain on its own. The input points, memory part queries and scene-agnostic queries are contextualized via a set of cross and self-attention operations, that iteratively update all queries and point features. Each of the contextualized queries predicts a segmentation mask proposal through inner-product with the contextualized point features, as well as an associated confidence score for the predicted part. These part segmentation proposals are matched to ground-truth part binary masks using Hungarian matching [23]. For the mask proposals that are matched to a ground-truth mask, we compute the segmentation loss, which is a per-point binary cross-entropy loss [35, 345]. We also supervise the confidence score of all queries. For implementation details, please see the Appendix, Section C.1.

Our modulator network resembles detection and segmentation transformers for 2D images [23, 34] where a set of learnable 1D vectors, termed parametric queries, iteratively cross-attend to input image features and self-attend among themselves to predict object segmentation masks in the input scene. The key difference between our modulator network and existing detection transformers are that **retrieved memory entity encodings are used as queries**, i.e., candidates for decoding parts in the input point cloud, alongside the standard set of scene-agnostic parametric queries, as shown in Figure 4.2. Additional differences are that we update the point features alongside the queries in the cross-attention layers, which we found helped performance.

4.3.1 Training

Analogical Networks aim to learn to associate parts in the retrieved memory graphs with parts in the input point cloud. We train our model in two stages to facilitate this fine-grained association learning:

1. Within-scene training: We apply two distinct augmentations (rotations and deformations [170]) to each training scene, and use one as the input scene and the other as the modulating memory, as shown in Figure 4.3. In this case, we **have access**

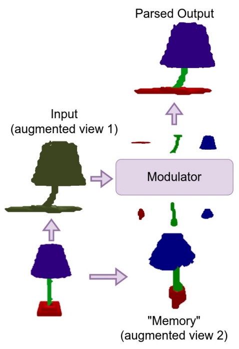


Figure 4.3: **Within-scene**

to ground-truth part associations between the parts of the memory and the input scene, which we use to supervise each memory part query to decode the corresponding part in the input cloud, and we do not use Hungarian matching. We call this within-scene training since both the input and memory come from the same scene instance.

2. Cross-scene training: During cross-scene training, the modulating memories are sampled from the top-K retrieved memories per input scene. We show in our experimental section that **accurate memory query to input part associations emerge** during Hungarian matching in cross-scene training, thanks to within-scene training. This is important, as often we cannot have fine-grained annotations of structure correspondence across scene exemplars.

In Analogical Networks, the retrieval process is not end-to-end differentiable with respect to the downstream scene parsing task. This is because i) we have no ground-truth annotations for retrieval, and ii) the retrieved memories are contextualized with the input through dense cross-attention operations which do not allow gradients to flow to the retriever's encoding parameters. We pre-train the encoder and modulator parameters independently of the retriever using within-scene training. We then use the modulator's encoder weights as the retriever's frozen encoder (adding a parameter-free average pooling layer on top). In the Appendix, we include pseudo-code for within-scene training in Algorithm 1 and cross-scene training in Algorithm 2.

4.4 Experiments

We test Analogical Networks on PartNet [243], an established benchmark for 3D object segmentation. PartNet contains 3D object instances from multiple object categories, annotated with parts in three different levels of granularity. We split PartNet object categories into base and novel categories. Our base categories are Chair, Display, Storage Furniture, Bottle, Clock, Door, Earphone, Faucet, Knife, Lamp, Trash Can, Vase and our novel categories are Table, Bed, Dishwasher and Refrigerator. We consider two experimental paradigms: **1. Many-shot:** For the exemplars of the base categories we consider the standard PartNet train/test splits. Our model and baselines are trained in the base category training sets, and tested on segmenting instances of the base categories in the test set. **2. Few-shot:** K labelled examples from each novel category are given and the model is tasked to segment new examples of these categories. This tests few-shot adaptation. We consider $K = 1$ and $K = 5$. We aggregate results across multiple K-shot learning tasks (episodes). We test two versions of our model: i) `AnalogicalNets single-mem`, which is Analogical Networks with a single modulating memory, and ii) `AnalogicalNets multi-mem`, with five modulating memories. Unless mentioned explicitly, Analogical Networks will imply the single-memory model.

Our experiments aim to answer the following questions:

1. How do Analogical Networks compare against parametric-alone state-of-the-art models in many-shot and few-shot 3D object segmentation?
2. How well do Analogical Networks adapt few-shot via memory expansion, without any weight update?
3. How do Analogical Networks compare against alternative memory-augmented networks where retrieved memory part encodings are attended to but not used as slots for decoding parts?
4. How well do Analogical Networks learn part-based associations across scenes without part association ground-truth or semantic label supervision?

Evaluation metrics We use the Adjusted Random Index (ARI) as our label-agnostic segmentation quality metric [281], which is a clustering score ranging from -1 (worst) to 1 (best). ARI calculates the similarity between two point clusterings while being invariant to the order of the cluster centers. We compute $100\times$ ARI using the publicly available implementation of [155]. We use mean Average Precision (mAP) per part [243] for semantic part instance segmentation and mean intersection over union (mIoU) for 3D point semantic segmentation.

Baselines We compare our model to existing models in the literature. We further compare against strong parametric baseline models we develop. Our parametric baselines already outperform all existing works in the many-shot settings. We consider the following baseline models:

- `PartNet` of [243] is a 3D segmentation network with the same backbone as our model. This model implements "queries" as a fixed number of sets of MLPs that operate over the encoded input point cloud. Each set contains one MLP for per-pixel confidence and one for part confidence. The same losses used by our model are used to supervise this baseline.
- `DETR3D` is a 3D segmentation transformer network with the same backbone as our model and similar segmentation prediction heads and losses, but without any memory retrieval, akin to the 3D equivalent of a state-of-the-art 2D image segmentor [9, 23]. We update the point features in the decoder layers, same as in our model, which we found to boost performance. Contrary to `DETR3D`, Analogical Networks attend to external memories.
- `PrototypicalNets` is an adaptation of the episodic prototypical networks for image classification of [319] to the task of 3D object part segmentation. Specifically, given a set of N labelled point clouds, we form average feature vectors for each semantic labelled part and use them as queries to segment points into corresponding part masks through inner-product decoding. Contrary to `PrototypicalNets`, Analogical Networks contextualize the memory queries and the input scene. `PrototypicalNets` require a retriever that knows the category label of the input scene, so it has access to privileged information. Without this assumption, we got very low performance from this model.

Ablations We compare our model to the following variants and ablative versions:

- `Re-DETR3D` (Retrieval-DETR3D) is a variant of Analogical Networks that attends to retrieved part memory encodings but does not use them to decode parts. Instead, all parts are decoded from the scene-agnostic queries. Different from Analogical Networks, analogies cannot emerge between a memory and the input scene since this model does not represent such correspondence explicitly, but only implicitly, in the attention operations.
- `AnalogicalNets w/o within-scene` is our `AnalogicalNets single-mem` model without any within-scene (pre)training.
- `AnalogicalNets OrclCategoryRetrv` is our `AnalogicalNets single-mem` model with a privileged retriever that has access to the ground-truth category label of

Method	Fine-tuned?	Novel Categories		Base Categories
		1-shot ARI (↑)	5-shot ARI (↑)	Many shot ARI (↑)
`PartNet`	✗	26.1	26.1	54.3
	✓	22.0 ± 0.90	25.9 ± 0.87	-
`DETR3D`	✗	30.4	30.4	**74.3**
	✓	39.4 ± 1.44	52.7 ± 1.44	-
`Re-DETR3D`	✗	38.2 ± 1.79	46.8 ± 0.66	**74.3**
	✓	46.5 ± 2.61	55.6 ± 1.82	-
`AnalogicalNets single-mem`	✗	$\mathbf{49.0 \pm 0.80}$	52.0 ± 1.11	72.5
	✓	$\mathbf{50.6 \pm 2.72}$	$\mathbf{57.0 \pm 1.33}$	-
`AnalogicalNets multi-mem`	✗	-	$\mathbf{52.1 \pm 0.75}$	74.2
	✓	-	56.4 ± 1.81	-
`AnalogicalNets OrclCategoryRetrv`	✗	51.2 ± 0.96	53.8 ± 1.03	75.6
	✓	52.6 ± 2.96	58.3 ± 1.36	-

Table 4.1: **Semantics-free 3D part segmentation performance** on the PartNet benchmark. Without any fine-tuning, Analogical Networks outperform `DETR3D` by more than 20% in the few-shot setup. Even upon fine-tuning, Analogical Networks outperform `DETR3D` by 4.3% ARI.

the input and only retrieves memories of the same category label.

4.4.1 Many and few-shot 3D object segmentation

The PartNet benchmark provides three levels of segmentation annotations per object instance where level 3 is the most fine-grained. We train and test our model and baselines on all three levels. We use a learnable level embedding as additional input for our baselines `PartNet` and `DETR3D`, as is usually the case in multi-task models. In the many-shot setting, we train our model and baselines jointly across all base categories and test them across all of them as well, using the standard PartNet train/test splits. For Analogical Networks and `Re-DETR3D`, all examples in the training set become part of their memory repository. In the few-shot setting, `PartNet` and `DETR3D` adapt by weight finetuning on the K-shot task. Analogical Networks and `Re-DETR3D` adapt in two ways: i) by expanding the memory of the model with the novel K-shot support examples, and ii) by further adapting the weights via fine-tuning to segment the K examples. In this case, the memory set is only the novel labelled support set instances. **The retriever does not have access to the object category information in any of the many-shot or few-shot settings** unless explicitly stated so.

Semantics-free part instance segmentation

We train all models and baselines for object part segmentation objectives, without any semantic label information, as described in Section 4.3. We show quantitative many-shot and few-shot part segmentation results in Table 4.1. For the few-shot setting, we show both fine-tuned and non-fine-tuned models. For the few-shot performance, we report mean and standard deviation over 10 tasks where we vary the K-shot support set. Our conclusions are as follows:

(i) Analogical Networks dramatically outperform `DETR3D` in few-shot part segmentation. While in the many-shot setting the two models have similar performance, when adapting few-shot to novel categories, Analogical Networks and all their variants dramatically outperform parametric alone `DETR3D`, both before and after fine-tuning.

(ii) Analogical networks can adapt few-shot simply by memory expansion, without weight updates. Indeed, the 5-shot performance of our model is close before and after fine-tuning in the novel categories (52.0% versus 57.0% ARI).

(iii) `AnalogicalNets multi-mem` outperform the single-memory version in many-shot learning with on par few-shot performance.

(iv) `Re-DETR3D` adapt few-shot better than `DETR3D`. Still, before weight fine-tuning in the few-shot test set, this memory-augmented variant exhibits worse performance than our single-memory model (46.8% versus 52.0% ARI).

(v) A retriever that better recognizes object categories would provide a performance boost, especially in the many-shot setting.

Emergent cross-scene correspondence

Analogical Networks can label the parts they segment with semantic categories by propagating semantic labels of the memory parts that were used to decode the input parts. We measure semantic instance segmentation performance for our model and baselines in Table 4.2. All variants of Analogical Networks are trained *without any semantic labelling objectives.* On the contrary, we train the baselines `DETR3D` and `PrototypicalNets` for *both object part segmentation and part labelling.* Analogical Networks predict semantic part labels only via propagation from memory part queries that decode parts in the scene, and does not produce any semantic labels for parts decoded by scene-agnostic queries, so by default it will make a mistake each

Method	Novel Categories 1-shot		Novel Categories 5-shot		Base Categories Many-shot	
	mIoU	mAP	mIoU	mAP	mIoU	mAP
`DETR3D`*	21.5	18.3	30.6	27.5	55.9	53.6
`AnalogicalNets single-mem w/o within-scene`	5.0	3.3	4.7	3.9	7.8	6.2
`AnalogicalNets single-mem`	20.4	18.2	26.0	25.0	44.3	42.0
`AnalogicalNets multi-mem`	-	-	27.8	25.8	49.2	47.9
`PrototypicalNets`*	27.5	-	29.0	-	30.0	-
`AnalogicalNets OrclCategoryRetrv`	26.2	25.2	30.2	30.2	50.6	48.7

Table 4.2: **Part Semantic and Part Instance Segmentation performance** on the PartNet benchmark. * indicates training with semantic labels.

time a parametric query is used. We found more than 80% of parts are decoded by memory part queries on average, while for `AnalogicalNets multi-mem` this ratio is 98%. For the few-shot settings in Table 4.2, all models are fine-tuned on the few given examples. Similar to our model, `PrototypicalNets` propagate semantic labels of the prototypical part features. We evaluate `PrototypicalNets` only for semantic segmentation since it cannot easily produce instance segments: if multiple part instances share the same semantic label, this model assumes they belong to the same semantic prototype. Our conclusions are as follows:

(i) Analogical Networks show very competitive semantic and instance segmentation accuracy via label propagation through memory part queries, despite having never seen semantic labels at training time. This shows **our model learns cross-scene part associations without any semantic information.**

(ii) `AnalogicalNets single-mem w/o within-scene` has much worse semantic segmentation performance which suggests **within-scene training much helps cross-scene associations to emerge without semantic information.**

(iii) `PrototypicalNets` achieves high 1-shot performance but does not scale with more data and is unable to handle both few-shot and many-shot settings efficiently.

(iv) A retriever that better recognizes object categories boosts performance of Analogical Networks over `DETR3D` in the few-shot settings.

Multi-category training helps few-shot adaptation in Analogical Networks

We compare our model and baselines on their ability to improve few-shot learning performance with more diverse training data in Table 4.3. We train each model under

Method	Fine-tuned?	Novel Categories 5-shot ARI (↑)	Chair ARI (↑)
`DETR3D` trained on "Chair"	✗	28.5	76.0
	✓	47.7 ± 2.31	-
`DETR3D`	✗	30.4	75.7
	✓	52.7 ± 1.44	-
Analogical Networks trained on "Chair"	✗	33.8 ± 0.89	**76.5**
	✓	50.4 ± 1.60	-
Analogical Networks	✗	$\mathbf{52.0 \pm 1.11}$	76.3
	✓	$\mathbf{57.0 \pm 1.33}$	-

Table 4.3: **Few-shot learning for single-category and multi-category trained models.** Analogical Networks learn better few-shot when trained across all categories, while `DETR3D` does not improve its few shot learning performance when trained across more categories.

two setups: i) training only on instances of "Chair", which is the most common category—approximately 40% of our training examples fall into this category—(refer to Table C.2 for statistics of the training data distribution) and, ii) training on "all" categories. We test the performance of each model on 5-shot learning. Our conclusions are as follows: **(i) Models trained on a single category usually fail to few-shot generalize to other classes** with or w/o fine-tuning, despite their strong performance on the training class. **(ii)** `DETR3D` does not improve in its ability to adapt few-shot with more diverse training data, in contrast to Analogical Networks.

In the Appendix, we show the effect of different retrieval mechanisms (C.2, C.3) and qualitative results on more benchmarks (C.4).

4.4.2 Limitations - Discussion

A set of future directions that are necessary for Analogical Networks to scale beyond segmentation of single-object scenes are the following: **(i)** The retriever in Analogical Networks operates currently over whole object memories and is not end-to-end differentiable with respect to the downstream task. Sub-object part-centric memory representations would permit fine-grained retrieval of visual memory scenes. We further plan to explore alternative supervision for the retriever module inspired by works in the language domain [130, 131]. **(ii)** Scaling Analogical Networks to segmentation of complete, multi-object 3D scenes in realistic home environments requires scaling up the size of memory collection. It would further necessitate bootstrapping fine-grained object part annotations, missing from 3D scene datasets [48], by transferring knowledge of object part compositions from PartNet. Such semi-supervised fine-grained scene parsing is an exciting avenue of future work. **(iii)** So far we have assumed the input to Analogical Networks to be a complete 3D point cloud. However, this is hardly ever the case in reality. Humans and machines need to make sense of single-view, incomplete and noisy observations. Extending Analogical Networks with generative heads that not only detect analogous parts in the input, but also inpaint or complete missing parts, is a direct avenue for future work.

4.5 Conclusion

We presented Analogical Networks, a semi-parametric model for associative 3D visual parsing that puts forward an analogical paradigm of corresponding input scenes to compositions and modifications of memory scenes and their labelled parts, instead of mapping scenes to segments directly. One-shot, few-shot or many-shot learning are treated uniformly in Analogical Networks, by conditioning to the appropriate set of memories, whether taken from a single, few or many memory exemplars, and inferring analogous parses. We showed Analogical Networks outperform SOTA parametric and meta-learning baselines in few-shot 3D object parsing. We further showed correspondences emerge across scenes without semantic supervision, as a by-product of the analogical inductive bias and our within-scene augmentation training. In his seminal work “the proactive brain” [12], Moshe Bar argues for the importance of analogies and associations in human reasoning—highlighting how associations of novel inputs to analogous representations in memory can drive perceptual inference. Analogical Networks operationalize these insights in a retrieval-augmented 3D parsing framework, with analogy formation between retrieved memory graphs and input scene segmentations.

Chapter 5

Energy-based Models are Zero-Shot Planners for Compositional Scene Rearrangement

In our first work in manipulation, we employ a factorized policy scheme that detects objects, forms an abstract representation, imagines a goal configuration and then executes. At the heart of this formulation lies our first generative model formulation, energy-based models for goal imagination. Energy-based models have general properties, such as joint optimization of multiple costs, that we will incorporate into our future chapters.

5.1 Introduction

We consider the scene arrangement task shown in Figure 5.1. Given a visual scene and an instruction regarding object spatial relations, the robot is tasked to rearrange the objects to their instructed configuration. Our focus is on strong generalization to longer instructions with novel predicate compositions, as well as to scene arrangements that involve novel objects and backgrounds.

We propose generating goal scene configurations corresponding to language instruc-

This chapter is based on the paper previously published at RSS 2023 [93]

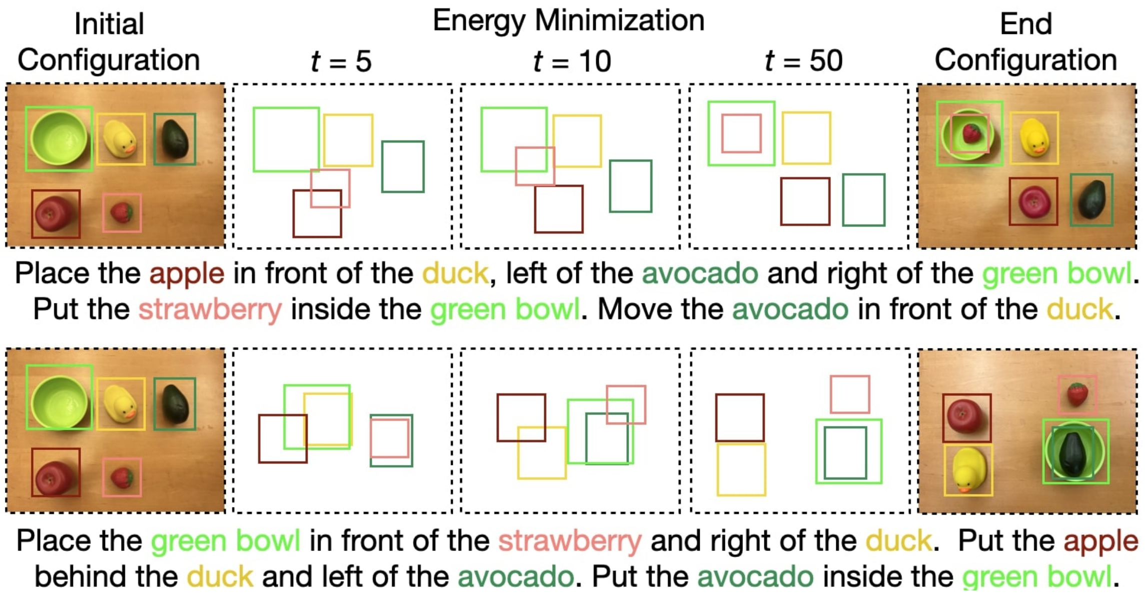

Initial Configuration
Energy Minimization
t = 5
t = 10
t = 50
End Configuration
Place the apple in front of the duck, left of the avocado and right of the green bowl. Put the strawberry inside the green bowl. Move the avocado in front of the duck.
Place the green bowl in front of the strawberry and right of the duck. Put the apple behind the duck and left of the avocado. Put the avocado inside the green bowl.

Figure 5.1: **Energy-based Models are Zero-Shot Planners for Compositional Scene Rearrangement**. We represent language concepts with energy functions over object locations and sizes. Gradient descent on the sum of energy functions, one per predicate in the instruction, iteratively updates the object spatial coordinates and generates a goal scene configuration that satisfies the instruction, if one exists.

tions by minimizing a composition of energy functions over object spatial locations, where each energy function corresponds to a language concept (predicate) in the instruction. We represent each language concept as an n-ary energy function over relative object poses and other static attributes, such as object size. We train these predicate energy functions to optimize object poses starting from randomly sampled object arrangements through Langevin dynamics minimization [70], using a handful of examples of visual scenes paired with single predicate captions. Energy functions can be binary for two-object concepts such as *left of* and *in front of*, or multi-ary for concepts that describe arrangements for sets of objects, such as *line* or *circle*. We show that gradient descent on the sum of predicate energy functions, each one involving different subsets of objects, generates a configuration that jointly satisfies all predicates, if this configuration exists, as shown in Figure 5.1.

We propose a robot learning framework that harnesses minimization of compositions of energy functions to generate instruction-compatible object configurations for robot scene rearrangement. A neural semantic parser is trained to map the input instruction to a set of predicates and corresponding energy functions, and the open-vocabulary visual-language grounding model proposed in Chapter 2 [135] grounds their arguments to objects in the scene, as shown in Figure 5.2. Gradient descent on the sum of energies with respect to the objects' spatial coordinates computes the final object locations that best satisfy the set of spatial constraints expressed in the instruction. Given the predicted object goal locations, we use vision-based pick-and-place policies that condition on the visual patch around the predicted pick and place locations to rearrange the objects [394]. We call our framework Scene Rearrangement via Energy Minimization (SREM).

We test SREM in scene rearrangement of tabletop environments on simulation benchmarks of previous works [311], as well as on new benchmarks we contribute that involve compositional instructions. We curate multiple train and test splits to test out-of-distribution generalization with respect to (i) longer instructions with more predicates, (ii) novel objects and (iii) novel background colors. We show SREM generalizes zero-shot to complex predicate compositions, such as *"put all red blocks in a circle in the plate"* **while trained from single predicate examples**, such as *"an apple inside the plate"* and *"a circle of blocks"*. We show SREM generalizes to real-world scene rearrangement without any fine-tuning, thanks to the object abstractions it operates on. We compare our model against state-of-the-art language-to-action policies [311] as well as Large Language Model planners [127] and show it dramatically outperforms both, especially for long complicated instructions. We ablate each component of our model and evaluate contributions of perception, semantic parsing, goal generation and low-level policy modules to performance.

In summary, our contributions are: **(i)** A novel energy-based object-centric planning framework for zero-shot compositional language-conditioned goal scene generation. **(ii)** A modular system for instruction-guided robot scene rearrangement that uses semantic parsers, vision-language grounding models, energy-based models for scene generation, and vision-based policies for object manipulation. **(iii)** A new instruction-guided scene rearrangement benchmark in simulation with compositional language instructions. **(iv)** Comparisons against state-of-the-art language-to-action policies and LLM planners, and extensive ablations.

Simulation and real-world robot execution videos, as well as our code are publicly available on our website: https://ebmplanner.github.io.

5.2 Related Work

Following instructions for rearranging scenes: Language is a natural means of communicating goals and can easily describe compositions of actions and arrangements [6, 42, 43], providing more versatile goal descriptions compared to supplying one or more goal images. The latter requires the task to be executed beforehand, which defeats the purpose of instruction [248, 268, 304, 358]. We group methods in the literature in the following broad categories:

- *End-to-end language to action policies* [207, 225, 311, 323] map instructions to actions or to object locations directly. We have found that these reactive policies, despite impressively effective within the training distribution, typically do not generalize to longer instructions, new object classes and attributes or novel backgrounds [207, 311].
- *Symbolic planners* such as PDDL (Planning Domain Definition Language) planners [156, 227, 238, 334] use predefined symbolic rules and known dynamics models, and infer discrete task plans given an instruction with lookahead logic search [88, 156, 156, 227, 238, 334]. Symbolic planners assume that each state of the world, scene goal and intermediate subgoal can be sufficiently represented in a logical form, using language predicates that describe object spatial relations. These methods predominantly rely on manually-specified symbolic transition rules, planning domains and grounding, which limits their applicability.
- *Large language models (LLMs)* map instructions to language subgoals [126, 127, 369, 414] or program policies [188] with appropriate plan-like prompts. The predicted subgoals interface with low-level short-term policies or skill controllers. LLMs trained from Internet-scale text have shown impressive zero-shot reasoning capabilities for a variety of downstream language tasks [22] when prompted appropriately, without any weight fine-tuning [202, 356]. The scene description is usually provided in a symbolic form as a list of objects present, predicted by open-vocabulary detectors [157]. Recent works of [188, 190] have also fed as input overhead pixel coordinates of objects to inform the LLM's predictions. The prompts for these methods need to be engineered per family of tasks. It is yet to be shown how the composition of spatial concept functions can emerge

in this way.

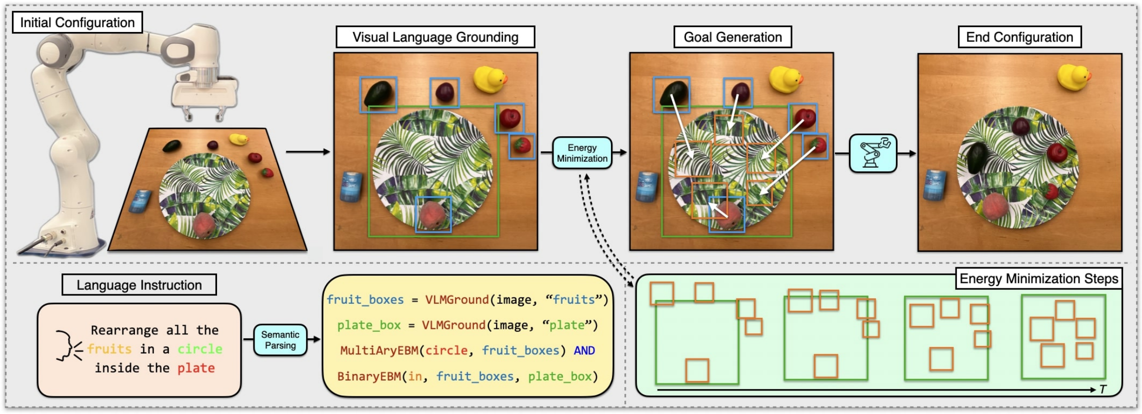
Language-conditioned scene generation: A large body of work has explored scene generation conditioned on text descriptions [153, 231, 279, 298, 386]. The work of [159] leverages web-scale pre-trained models [113, 276, 280] to generate segmentation masks for each object in the generated goal image. Given an input image, their method generates a text prompt using a captioning model and feeds it to a generative model that outputs a goal image, which is then further parsed into segmentation masks. However, the prompt is limited to contain only names of objects and there is no explicit language-guided spatial reasoning. In this work, we seek to make scene generation useful as goal imagination for robotic spatial reasoning and instruction following. Instead of generating pixel-accurate images, we generate object configurations by abstracting the appearance of object entities. We show this abstraction suffices for a great number of diverse scene rearrangement tasks.

Energy-based models: Our work builds upon existing work on energy-based models (EBMs) [67, 68, 70, 99, 201, 244]. Most similar to our work is that of [244], which generates and detects spatial concepts with EBMs on images with dots, and [68, 201], which demonstrates composability of image-centric EBMs for generating face images and images from CLEVR dataset [152]. In this work, we demonstrate zero-shot composability of EBMs over object poses instead of images, and showcase their applicability on spatial reasoning and instruction following for robotic scene rearrangement.

5.3 Method

The architecture of SREM is shown in Figure 5.2. The model takes as input an RGB-D image of the scene and a language instruction. A semantic parser maps the instruction to a set of spatial predicate energy functions and corresponding referential expressions for their object arguments. An open-vocabulary visual detector grounds the arguments of each energy function to actual objects in the scene. The goal object locations are predicted via gradient descent on the sum of energy functions. Lastly, short-term vision-based pick-and-place policies move the objects to their inferred goal locations. Below, we describe each component in detail.

A library of energy-based models for spatial concepts In our work, a



```
Rearrange all the
fruits in a circle
inside the plate
```

```
fruit_boxes = VLMGround(image, "fruits")
plate_box = VLMGround(image, "plate")
MultiAryEBM(circle, fruit_boxes) AND
BinaryEBM(in, fruit_boxes, plate_box)
```

Figure 5.2: **Scene rearrangement through energy minimization.** Given an image and a language instruction, a semantic parser maps the language into a set of energy functions (`BinaryEBM`, `MultiAryEBM`), one for each spatial predicate in the instruction, and calls to an open-vocabulary visual language grounder (`VLMGround`) to localize the object arguments of each energy function mentioned in the instruction, here "fruits" and "plate". Gradient descent on the sum of energy functions with respect to object spatial coordinates generates the goal scene configuration. Vision-based neural policies condition on the predicted pick and place visual image crops and predict accurate pick and place locations to manipulate the objects.

spatial predicate is represented by an energy-based model (EBM) that takes as input x the set of objects that participate in the spatial predicate and maps them to a scalar energy value $E_\theta(x)$. An EBM defines a distribution over configurations x that satisfy its concept through the Boltzmann distribution $p_\theta(x) \propto e^{-E_\theta(x)}$. Low-energy configurations imply satisfaction of the language concept and have high probability. An example of the spatial concept can be generated by optimizing for a low-energy configuration through gradient descent on (part of) the input x. We represent each object entity by its 2D overhead centroid coordinates and box size. During gradient descent, we only update the center coordinates and leave box sizes fixed. We consider both binary spatial concepts (*in*, *left of*, *right of*, *in front of*, *behind*) as well as multi-ary spatial concepts (*circle*, *line*).

Using an EBM, we can sample configurations from p_θ, by starting from an initial configuration x^0 and refining it using Langevin Dynamics [357]:

$$x^{k+1} = x^k - \lambda \nabla_x E_\theta(x^k) + \epsilon^k z^k, \tag{5.1}$$

where z^k is random noise, λ is an update rate hyperparameter and ϵ^k is a time-dependent hyperparameter that monotonically decreases as k increases. The role of z^k and decreasing ϵ^k is to induce noise in optimization and promote exploration, similar to Simulated Annealing [173]. After K iterations, we obtain $x^- = x^K$. During training, we iterate over Equation 5.1 $K = 30$ times, using $\lambda = 1$ and $\epsilon_k = 5e - 3$. During inference, we find that iterating for more, e.g. $K = 50$ often leads to better solution. In this case we also linearly decay ϵ_k to 0 for $k > 30$.

We learn the parameters θ of our EBM using a contrastive divergence loss that penalizes energies of examples sampled by the model being lower than energies of ground-truth configuration:

$$\mathcal{L} = \mathbb{E}_{x^+ \sim p_D} E_\theta(x^+) - \mathbb{E}_{x^- \sim p_\theta} E_\theta(x^-), \tag{5.2}$$

where x^+ a sample from the data distribution p_D and x^- a sample drawn from the learned distribution p_θ. We additionally use the KL-loss and the L2 regularization proposed in [70] for stable training. At test time, compositions of concepts can be created by simply summing energies of individual constituent concept, as shown in Figures 5.1 and 5.2.

We implement two sets of EBMs, a BinaryEBM and a MultiAryEBM for binary (e.g., *left of*) and multi-ary (e.g., *circle*) language concepts, respectively. The BinaryEBM expects two object arguments, each represented by its bounding box. We convert the object bounding box to (top-left corner, bottom-right corner) representation. Then we compute the difference between all corners of the two object arguments and concatenate and feed to a multi-layer perceptron (MLP) that outputs a scalar energy value. Note that the energy function only depends on the relative arrangement of the two objects, not their absolute locations. The MultiAryEBM is used for order-invariant concepts of multiple entities, such as shapes. The input is a set of objects, each represented as a point (box center). We subtract the centroid of the configuration from each point and then featurize each object using an MLP. We feed this set of object features to a sequence of four attention layers [342] for contextualization. The refined features are averaged into an 1D vector which is then mapped to a scalar energy using an MLP. We train a separate EBM for each language concept in our vocabulary using corresponding annotated scenes in given demonstrations. Note that annotated scenes suffice to train the energy functions, kinesthetic demonstrations are not necessary, and in practice each EBM can be trained within a few minutes.

Semantic parsing of instructions into spatial concepts and their arguments. Our parser maps language instructions to instantiations of energy-based models and their arguments. It is a Sequence-to-Tree model [63] with a copying mechanism [101] which allows it to handle a larger vocabulary than the one seen during training. The input to the model is a natural language instruction and the output is a tree. Each tree node is an operation. The three operations supported are i) `BinaryEBM` which calls a BinaryEBM from our library, ii) `MultiAryEBM` and iii) `VLMGround` which calls the visual-language grounding module. Each node has a pointer to the arguments of the operation, language concepts for EBM calls, e.g., *behind*, and noun phrases for grounding model calls, e.g., *"the green cube"*. Nodes in the parsing tree may also have children nodes, which imply nested execution of the corresponding operations. The input utterance is encoded using a pre-trained RoBERTa encoder [212], giving a sequence of contextualized word embeddings and a global representation of the full utterance. Then, a decoder is iteratively employed to i) decode an operation, ii) condition on this operation to decode or copy the arguments for this operation, iii) add one (or more) children node(s). For example, the instruction *"a circle of cubes inside the plate"* is mapped to a sum of energy functions where each object of the multi-ary concept *circle* participates in the constraining binary concept *in*:

$$\begin{aligned} E^{total} = \;& \texttt{MultiAryEBM}(circle, \texttt{VLMGround}(\text{“}cubes\text{”})) \\ & + \textstyle\sum_i \texttt{BinaryEBM}(in, x_i, \texttt{VLMGround}(\text{“}plate\text{”})), \\ & x_i \in \texttt{VLMGround}(\text{“}cubes\text{”}). \end{aligned} \tag{5.3}$$

We train our semantic parser on the instructions of all training demonstrations of all tasks jointly, as well as on synthesized instructions paired with programs, each with 1-7 predicates, that we generate by sampling from a grammar, similar to previous works [233, 354].

We ground noun phrases predicted by our parser with an off-the-shelf language grounding model [135], which operates as an open-vocabulary detector. The input is the noun phrase, e.g., *"the blue cube"* and the image, while the output is the boxes of all object instances that match the noun phrase. The open-vocabulary detector has been pre-trained for object detection and referential grounding on MS COCO [193], Flickr30k [265] and Visual Genome [175]. We finetune the publicly available code of [135] on our training data of all tasks jointly.

Short-term vision-based manipulation skills We use short-term manipulation policies built upon Transporter Networks [394] to move the obejcts to their predicted locations. Transporter Networks take as input one or more RGB-D images, reproject them to the overhead birds-eye-view, and predict two robot gripper poses: i) a pick pose and ii) a pick-conditioned placement pose. These networks can model any behaviour that can be effectively represented as two consecutive poses for the robot gripper, such as pushing, sweeping, rearranging ropes, folding, and so on – for more details please refer to [394].

We modify Transporter Networks to take as input a small image RGB-D patch, instead of a complete image view. Specifically, we consider as input the image patches around the object pick and object goal locations predicted by our visual grounding and energy-based minimization modules respectively. In this way, the low-level policies know roughly what to pick and where to place it, and only locally optimize over the best pick location, as well as the gripper's relative rotation, within an object of interest, or placement location, at a particular part of the scene, respectively. We show in our ablations (Table 5.7) that using learning-based pick-and-place policies helps performance, even if the search space is limited thanks to grounding and goal imagination. We train Transporter Networks from scratch on all our pick-and-place demonstration datasets jointly.

Termination of execution: SREM generates a goal scene by optimizing the relative poses of the objects mentioned in the instruction. We estimate how many objects should be moved by comparing the detected bounding box (by the language grounding model) and the optimized bounding box (by the EBM). For non-compositional tasks that involve binary concepts, we inject the prior that one object is fixed. Then we take as many actions as the number of objects the EBM moved.

Closed-loop execution: SREM first generates a goal scene from the input instruction and then executes it. After execution, we re-detect all relevant objects using our VLM-grounder module to check if they are close to their predicted goal locations. If the re-detected object's bounding box and initially predicted goal bounding box intersect over a certain IoU threshold, we consider the goal to be successfully executed. If we fail to reach the goal, we call again our vision based policies using the current scene configuration. Comparing the post-execution object configuration with the initially imagined goal scene allows to track progress and estimate goal completion as we show in the experimental section.

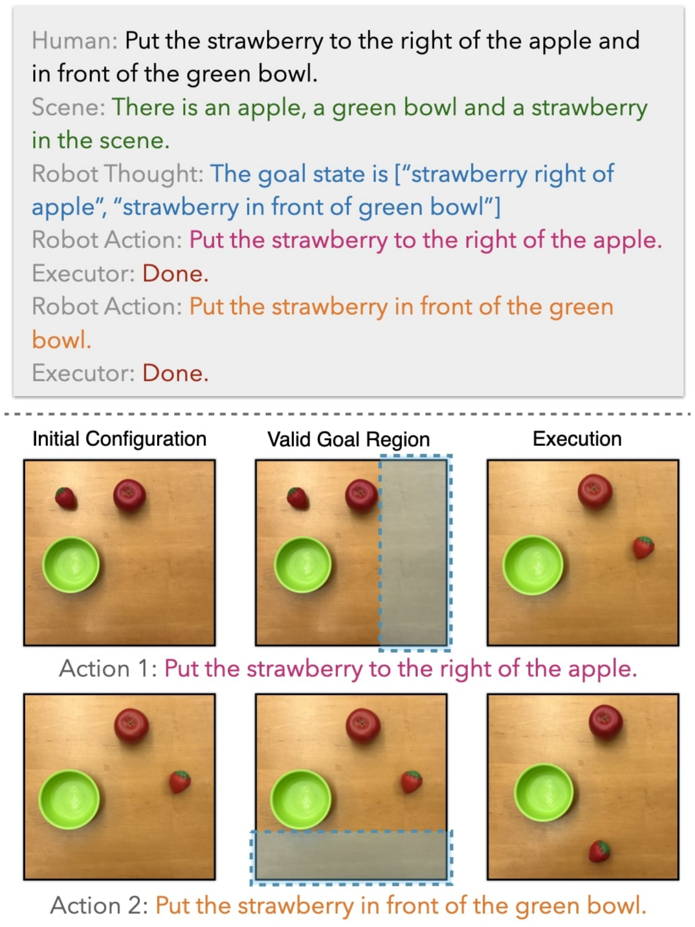
5.4 Experiments

We test SREM in its ability to follow language instructions for rearrangement of tabletop scenes in simulation and in the real world. We compare our model against LLM planners [127] and end-to-end language-to-action policies [311]. Our experiments aim to answer the following questions:

1. How does SREM compare to LLM planners in predicting scene configurations from instructions? (Section 5.4.1)
2. How does SREM compare to state-of-the-art language-to-action policies for rearranging scenes? How does their relative performance change with varying instruction length and varying amount of training data? (Section 5.4.2)
3. How does SREM generalize to novel objects, object colors and background colors, compared to an end-to-end language-to-action model? (Section 5.4.3)
4. How much do different modules of our framework contribute to performance? (Section 5.4.4)

Benchmarks: Existing language-conditioned manipulation benchmarks are usually dominated by a single spatial concept like "inside" [311]. To better illustrate the compositionality of spatial concepts, we introduce the following set of benchmarks, implemented with PyBullet:

- **spatial-relations**, containing single pick-and-place instructions with referential expressions in cluttered scenes with distractors, e.g. *"Put the cyan cube above the red cylinder"*. We consider the relations *left of*, *right of*, *in front of*, *behind.*
- **comp-one-step**, containing compositional instructions with referential expressions in cluttered scenes with distractors that require one object to be re-located to a particular location, e.g. *"put the red bowl to the right of the yellow cube,*


Human: Put the strawberry to the right of the apple and in front of the green bowl.
Scene: There is an apple, a green bowl and a strawberry in the scene.
Robot Thought: The goal state is ["strawberry right of apple", "strawberry in front of green bowl"]
Robot Action: Put the strawberry to the right of the apple.
Executor: Done.
Robot Action: Put the strawberry in front of the green bowl.
Executor: Done.

Initial Configuration
Valid Goal Region
Execution
Action 1: Put the strawberry to the right of the apple.
Action 2: Put the strawberry in front of the green bowl.

Figure 5.3: **Planning in language space with Large Language Models (LLMs).** LLM Planners predict language subgoals that decompose the initial instruction to simpler-to-execute subtasks. Predicted language subgoals are fed to reactive language-to-action policies for execution. In cases where concept intersection is needed, the predicted sequential language subgoal decomposition of instructions can fail. Here, the LLM predicts the first subgoal of putting the strawberry to the right of the apple. The reactive policy can succeed if it places the strawberry anywhere within the shaded region. During execution of the next issued language subgoal of putting the strawberry in front of the bowl, the policy violates the first constraint. Placing the strawberry in the intersection of the two shaded regions may not be achieved by decomposing the two predicates sequentially, as opposed to composing them. Then the burden of handling the compositional instruction is outsourced to the language-to-action policy, which often fails to generalize. Instead, SREM directly addresses compositionality of multiple spatial language predicates.

to the left of the red cylinder, and above blue cylinder".

- **comp-group**, containing compositional instructions with referential expressions in cluttered scenes with distractors that require multiple objects to be re-located, e.g., *"put the grey bowl above the brown cylinder, put the yellow cube to the right of the blue ring, and put the blue ring below the grey bowl"*.
- **shapes**, containing instructions for making multi-entity shapes (circles and lines), e.g. *"rearrange all red cubes in a circle"*.

We further evaluate our model and baselines on four tasks from the CLIPort benchmark [311], namely **put-block-in-bowls**, **pack-google objects-seq**, **pack-google objects-group** and **assemble-kits-seq**.

For all tasks we train on either 10 or 100 demos and use the same demos to train all our modules, as discussed in Section 5.3. We test on 50 episodes per task, where we vary the instruction and the initial configuration of objects. For **spatial-relations** and **shapes** each concept corresponds to a task, while the composition benchmarks correspond to one task each.

Baselines: We compare SREM to the following baselines:

- CLIPort [311], a model that takes as input an overhead RGB-D image and an instruction and uses pre-trained CLIP language and image encoders to featurize the instruction and RGB image, respectively; then fuses these with depth features to predict pick-and-place actions using the action parametrization of Transporter Networks [394]. The model capitalizes on language-vision associations learnt by the CLIP encoders. We use the publicly available code of [311]. We train one CLIPort model on all tasks of each benchmark, e.g., one model for **spatial-relations**, a different for **comp-group** etc. Note that the original CLIPort implementation assumes access to oracle success/failure information based on which the model can retry the task for a fixed budget of steps or stop the execution if oracle confirms that the task is completed. We evaluate the CLIPort model without this oracle retry but still with oracle information of how many minimum steps it needs to take to complete the task, so we force CLIPort to take exactly that number of actions.
- LLMplanner, inspired by [127], an instruction-following scene-rearrangement model that uses an LLM to predict a sequence of subgoals in language form,

e.g. *"pick the red cube and place it to the right of the blue bowl"*. The generated language subgoals are fed as input to language-to-action policies, such as CLIPort. Scene state description is provided as a list of objects in the scene. LLMplanner does not finetune the LLM but instead uses appropriate prompts so that the LLM adapts its behavior in-context and generates similar statements. The prompts include various previous successful interactions between a human user and the model. We design suitable prompts for our introduced benchmarks and use the LLM to decompose a long instruction into simpler ones (see Figure 5.3 for an example). Then, we feed each generated instruction to a CLIPort model, trained as described earlier. Lastly, for tabletop manipulation tasks in simulation, the LLMPlanner of [127] assumes access to an oracle success/failure detector. The difference in our implementation is that we do not assume any success detector. The execution terminates when all language subgoals have been fed to and handled by CLIPort.

Note that LLMplanner boils down to CLIPort for non-compositional instructions. As such, we compare with LLMplanner only on **comp-one-step** and **comp-group**, both in simulation and real world.

Evaluation Metrics: We use the following two evaluation metrics: (i) **Task Progress (TP)** [394] is the percentage of the referred objects placed in their goal location, e.g. $4/5 = 80.0\%$ for rearranging 4 out of 5 objects specified in the instruction. (ii) **Task Completion (TC)** rewards the model only if the full rearrangement is complete. For the introduced benchmarks we have oracle reward functions that evaluate whether the task constraints are satisfied.

5.4.1 Spatial reasoning for scene rearrangement with oracle perception and control

In this section, we compare spatial reasoning for predicting compositional scene subgoals in a language space versus in an abstract visually grounded space. In this section, to isolate this reasoning ability from nuisance factors of visually localizing the objects and picking them up effectively, we consider **oracle object detection, referential grounding and low-level pick-and-place policies.** Specifically, we carry out inferred language subgoals from LLMplanner using oracle controllers that

	comp-one-step		**comp-group**	
Method	TP	TC	TP	TC
LLMplanner w/ oracle	82.0	59.0	75.3	29.0
SREM w/ oracle	**90.8**	**76.0**	**88.7**	**62.0**

Table 5.1: **Evaluation of SREM and LLMplanner with oracle perception and oracle low-level execution policies** on compositional spatial arrangement tasks. We report Task Progress (TP) and Task Completion (TC).

	left-seen-colors		**left-unseen-colors**		**right-seen-colors**		**right-unseen-colors**	
Method	10 demos	100 demos	10 demos	100 demos	10 demos	100 demos	10 demos	100 demos
CLIPort	13.0	44.0	9.0	33.0	29.0	43.0	28.0	44.0
SREM	**95.0**	**95.0**	**93.0**	**94.0**	**89.0**	**92.0**	**93.0**	**96.0**

	behind-seen-colors		**behind-unseen-colors**		**front-seen-colors**		**front-unseen-colors**	
Method	10	100	10	100	10	100	10	100
CLIPort	24.0	45.0	22.0	51.0	23.0	55.0	13.0	40.0
SREM	**87.0**	**87.0**	**89.0**	**90.0**	**89.0**	**90.0**	**88.0**	**89.0**

	circle-seen-colors		**circle-unseen-colors**		**line-seen-colors**		**line-unseen-colors**	
Method	10 demos	100 demos	10 demos	100 demos	10 demos	100 demos	10 demos	100 demos
CLIPort	34.1	61.5	31.2	55.6	48.6	88.2	48.6	88.5
SREM	**91.3**	**91.5**	**90.2**	**91.2**	**98.1**	**99.0**	**98.4**	**99.4**

Table 5.2: **Evaluation (TP) of SREM and CLIPort on spatial-relations and shapes in simulation.**

relocate an object in the scene such that it satisfies the predicted subgoals. Note that SREM relies on pick-and-place policies that are not language-conditioned, while LLMplanner relies on language-conditioned policies for object re-location. Thus, the oracle control assumption is less realistic in the latter case. We forego this difference for the sake of comparison.

We show quantitative results of SREM and LLMplanner on the **comp-one-step** and **comp-group** benchmarks in Table 5.1. Our model outperforms LLMplanner and the performance gap is larger in more complex instructions. To elucidate why an abstract visual space may be preferable for planning, we visualize steps of energy minimization for different instructions in Figure 5.1 and steps of the execution of the LLM prompted by us to the best of our capability in Figure 5.3. We can see that SREM trained on single-predicate scenes shows remarkable composability in case of

	comp-one-step seen-colors		**comp-one-step unseen-colors**		**comp-group seen-colors**		**comp-group unseen-colors**	
Method	10	100	10	100	10	100	10	100
Initial (no movement)	0.0	0.0	0.0	0.0	31.7	31.7	31.8	31.8
CLIPort (zero-shot)	9.0	12.0	7.0	12.0	37.4	37.5	32.6	38.4
CLIPort	13.0	15.0	14.0	9.0	38.2	38.5	34.7	40.9
LLMplanner	51.2	53.2	49.4	53.5	38.6	39.0	37.1	39.0
SREM (zero-shot)	90.0	91.0	92.7	90.3	77.2	77.4	77.7	78.4
SREM (zero-shot + closed-loop)	**91.6**	**92.0**	**92.9**	**91.4**	**80.8**	**81.6**	**81.1**	**82.4**

Table 5.3: **Evaluation (TP) of SREM, CLIPort and LLMplanner on compositional tasks**. SREM is trained only on atomic relations and tested zero-shot on tasks with compositions of spatial relations which involve moving one (**comp-one-step**) or multiple (**comp-group**) objects to satisfy all constraints specified by the language. Some language constraints are satisfied already in the initial configuration and the Initial model captures that.

	put-block-in-bowl seen-colors		**put-block-in-bowl unseen-colors**		**packing-google-objects seq-seen-objects**		**packing-google-objects seq-unseen-objects**	
Method	10 demos	100 demos	10 demos	100 demos	10 demos	100 demos	10 demos	100 demos
CLIPort	31.0	82.1	4.8	17.6	34.8	54.7	27.2	56.4
SREM	**84.3**	**93.8**	**89.0**	**95.3**	**86.8**	**94.8**	**88.0**	**92.9**

	packing-google-objects group-seen-objects		**packing-google-objects group-unseen-objects**		**assembling-kits seq-seen-colors**		**assembling-kits seq-unseen-colors**	
Method	10	100	10	100	10	100	10	100
CLIPort	33.5	61.2	32.2	70.0	38.0	**62.6**	36.8	**51.0**
SREM	**86.1**	**76.8**	**87.2**	**79.6**	**38.4**	42.0	**40.8**	44.0

Table 5.4: **Evaluation (TP) of SREM and CLIPort on CLIPort benchmark in simulation.**

multiple predicates. Language planning on the other hand suffers from the ambiguity of translating geometric concepts to language and vice versa: step-by-step execution of language subgoals does not suffice for the composition of the two subgoals to emerge (Figure 5.3).

5.4.2 Spatial scene rearrangement

Simulation: In this section, we compare our model and the baselines in the task of instruction-guided scene rearrangement. We first show results on **spatial-relations** and **shapes** in Table 5.2. We largely outperform CLIPort, especially when less training demos are considered.

To evaluate generalization on longer instructions at test time, we show quantitative results in Table 5.3 for the benchmarks of **comp-one-step** and **comp-group**. We compare our model with CLIPort trained on atomic spatial relations and zero-shot evaluated on compositional benchmarks. We further fine-tune CLIPort on demos from the compositional benchmarks. SREM is not trained on these benchmarks, because the energy functions are already composable, meaning that we can jointly optimize over an arbitrary number of constraints by simply summing the different energy terms. Under all different settings, we significantly outperform all variants of CLIPort and LLMplanner. We also observe that closed-loop execution boosts our performance further.

We additionally show results on the CLIPort benchmark in Table 5.4. We largely outperform CLIPort on almost all tested tasks. Margins are significantly larger when i) less demos are used and ii) the robot has to interact with objects of unseen colors or classes. Most of the failure cases for our model are due to the language grounding mistakes - in particular for assemble-kits-seq we find that the grounder gets confused between letters and letter holes.

Real World: We test our model on a 7-DoF Franka Emika robot, equipped with a parallel jaw gripper and a top-down Azure Kinect RGB-D camera. We do not perform any real-world finetuning. Our test set contains 10 language-guided tabletop manipulation tasks per setting (Comp-one-step, Comp-group, Circles, Lines). We show quantitative results in Table 5.6. SREM generalizes to the real world without any real-world training or adaptation thanks to the open-vocabulary detector trained on real-world images, as well as the object abstractions in the predicate EBMs and low-level policy modules. We encourage readers to refer to our supplementary video and our website for more detailed results.

5.4.3 Generalization analysis

We conduct controlled studies of our model's generalization across three axes: a) **novel colors**: we train the models with objects of 7 different colors and evaluate them on objects of 4 unseen colors; b) **novel background colors**: we train all models on black-colored tables and evaluate on tables of randomly sampled RGB

		spatial-relations		**composition**	
Novel attribute	Model	10 demos	100 demos	10 demos	100 demos
None	CLIPort	22.0	47.0	25.6	26.8
	SREM	90.0	91.0	83.6	84.2
Color	CLIPort	18.0	39.0	25.1	24.5
	SREM	87.0	85.0	86.5	84.0
Background	CLIPort	10.0	20.0	23.7	23.2
	SREM	79.0	68.0	77.0	72.0
Objects	CLIPort	17.0	19.0	24.5	24.8
	SREM	86.0	86.0	80.9	81.5

Table 5.5: **Generalization experiments of SREM and CLIPort in manipulation tasks in simulation (metric is TP).**

Method	**comp-one-step**	**comp-group**	**circles**	**lines**
CLIPort	13.1	22.9	34.0	46.0
LLMplanner	39.5	25.9	-	-
SREM	**85.6**	**75.8**	**94.0**	**90.0**

Table 5.6: **Real-world evaluation (TP) of SREM**

Method	Accuracy
SREM	77.2
SREM w/o goal generation	42.1
SREM w/o learnable policies	61.2
SREM w/ oracle language grounding	82.3
SREM w/ everything oracle except goal	88.3

Table 5.7: **Ablations of SREM on the benchmark comp-group-seen-colors (metric is TP).**

colors; c) **novel objects**: we train the models on objects of 4 classes and evaluate on rearrangement of 11 novel classes. In each of these settings, we only change one attribute (i.e. object color, background color or object instance) while keeping everything else constant.

We evaluate our model and CLIPort trained on 10 or 100 demos per task on **spatial-relations** (average performance over all tasks) and **composition** (average performance over all tasks from **comp-one-step** and **comp-group**). The results are summarized in Table-5.5. We observe that our model maintains high performance across all axes of generalization, independently of the number of training demos.

Our model's generalization capabilities rely on the open-vocabulary detector and the fact that EBMs and transporter-based low-level execution policy operate on abstracted space in a modular fashion. While CLIPort models can also generalize to novel scenarios by leveraging the CLIP model, the action prediction and perception are completely entangled and hence even if CLIP manages to identify the right objects based on the language, it has trouble predicting the correct pick and place locations.

5.4.4 Ablations

We show an error analysis of our model in Table-5.7. First, we remove the goal generation from SREM (SREM w/o goal generation) by conditioning the place network on the language input instead of the EBM-generated goal image, while keeping the pick network and object grounders identical. We observe a drop of 35.1% in accuracy, underscoring the importance of goal generation. We then remove our executor policy (SREM w/o learnable policies) and instead randomly select pick/place locations inside the bounding box of the relevant object. This results in a drop of 16%, showing the importance of robust low-level policies. We do not remove the grounder and parser since they are necessary for goal generation. We then experiment with oracle visual language grounder (SREM w/ oracle language grounding) that perfectly detects the objects mentioned in the sentence, which results in a performance gain of 5.1%. We finally evaluate with perfect grounding, language parsing and low-level execution (SREM w/ everything oracle except goal) to test the error rate of our goal generator. We obtain an 88.3% accuracy, thus concluding that our goal generator fails in 11.7% cases.

5.4.5 Limitations

Our model presently has the following two limitations: First, it predicts the goal object scene configuration but does not have any knowledge regarding temporal ordering constraints on object manipulation execution implied by physics. For example, our model can predict a stack of multiple objects on top of one another but cannot suggest which object needs to be moved first. One solution to this problem is to heuristically pick the order based on objects that are closer to the floor in the predicted scene configuration. However, more explicit encoding of physics priors are important to also identify if the generated configuration is stable or not. A promising direction is to model physics-based constraints as additional energy constraints, and obtain optimization gradients by leveraging either differentiable physics simulators [129, 274, 365] or learned dynamics models [187, 264, 363]. Second, our EBMs are currently parametrized by object locations and sizes, but different tasks need different abstractions. Manipulation of articulated objects, fluids, deformable objects or granular materials, would require finer-grained parametrization in both space and time. Furthermore, even for rigid objects, many tasks would require finer in-space parametrization, e.g., it would be useful to know a set of points in the perimeter of a plate as opposed to solely representing its bounding box for accurately placing things inside it. Considering EBMs over keypoint or object part graphs [232, 315] is a direct avenue for future work.

5.5 Conclusion

We introduce SREM, a modular robot learning framework for instruction-guided scene rearrangement that maps instructions to object scene configurations via compositional energy minimization over object spatial coordinates. We test our model in diverse tabletop manipulation tasks in simulation and in the real world. Our model outperforms state-of-the-art end-to-end language-to-action policies, and LLM-based instruction following methods both in in- and out-of-distribution settings, and across varying amount of supervision. We contribute a new scene rearrangement benchmark that contains more compositional language instructions than previous works, which we make publicly available to the community. Our work shows that a handful of visually-grounded examples suffice to learn energy-based spatial language concepts that can be composed to infer novel instructed scene arrangements, in long and complex compositional instructions.

Chapter 6

Act3D: 3D Feature Field Transformers for Multi-Task Robotic Manipulation

Act3D is 3D policy but not a generative one. However, it serves a valuable transition into unifying the input and output space in manipulation. While the previous chapter adopted Detection Transformers as a tool, Act3D directly builds a Detection Transformer-based architecture for manipulation.

6.1 Introduction

Solutions to many robot manipulation tasks can be modeled as a sequence of 6-DoF end-effector poses (3D position and orientation). Many recent methods train neural manipulation policies to predict 3D end-effector pose sequences directly from 2D images using supervision from demonstrations [19, 103, 140, 199, 312, 315]. These methods are typically sample inefficient: they often require many trajectories to handle minor scene changes at test time and cannot easily generalize across camera viewpoints and environments, as mentioned in the respective papers and shown in our experiments.

This chapter is based on the paper previously published at CoRL 2023 [91]

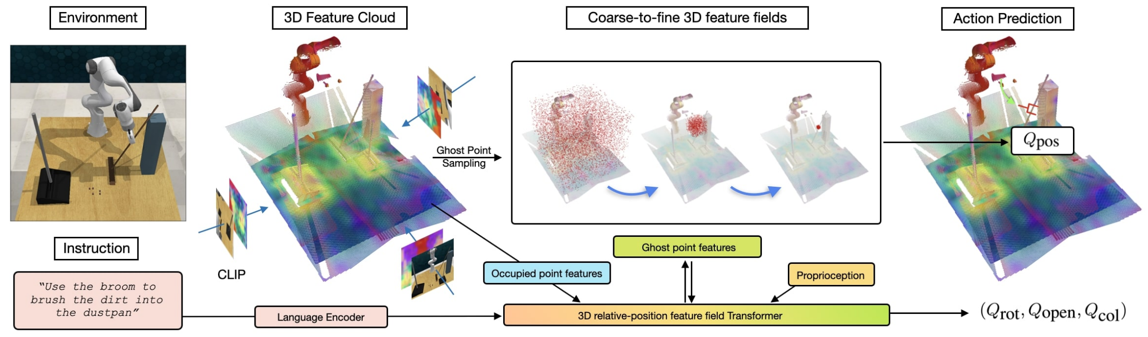

Environment
3D Feature Cloud
Coarse-to-fine 3D feature fields
Action Prediction
Ghost Point Sampling
Q_{pos}
Instruction
"Use the broom to brush the dirt into the dustpan"
CLIP
Ghost point features
Occupied point features
Proprioception
Language Encoder
3D relative-position feature field Transformer
$(Q_{\text{rot}}, Q_{\text{open}}, Q_{\text{col}})$

Figure 6.1: **Act3D** is a language-conditioned robot action transformer that learns 3D scene feature fields of arbitrary spatial resolution via recurrent coarse-to-fine 3D point sampling and featurization using relative-position attentions. Act3D featurizes multi-view RGB images with a pre-trained 2D CLIP backbone and lifts them in 3D using sensed depth. It predicts 3D location of the end-effector using classification of the 3D points of the robot's workspace, which preserves spatial equivariance of the scene to action mapping.

For a robot policy to generalize under translations, rotations, or camera view changes, it needs to be spatially equivariant [413], that is, to map 3D translations and rotations of the input visual scene to similar 3D translations and rotations for the robot's end-effector. Spatial equivariance requires predicting 3D end-effector locations through 2D or 3D action maps, depending on the action space considered, instead of regressing action locations from holistic scene or image features. Transporter networks [396] introduced a spatial equivariant architecture for 4-DoF robot manipulation: they re-project RGB-D input images to a top-down image and predict robot end-effector 2D translations through a top-down 2D action map. They showed better generalization with fewer training demonstrations than prior works. However, they are limited to top-down 2D worlds and 4-DoF manipulation tasks. This begs the question: how can we extend spatial equivariance in action prediction to general 6-DoF manipulation?

Developing spatially equivariant 6-DOF manipulation policies requires predicting 3D action maps by classifying 3D points in the robot's workspace as candidates for future 3D locations for the robot's end-effector. Predicting high-resolution 3D action maps, necessary for fine-grained manipulation tasks, poses a computational challenge over their 2D counterparts due to the extra spatial dimension. Voxelizing the robot's

3D workspace and featurizing the 3D voxels at high resolution is computationally demanding [338]. The next end-effector pose might be anywhere in free space, which prevents the use of sparse 3D convolutions [40, 96] to selectively featurize only part of the 3D free space. To address this, recent work of PerAct [312] featurizes 3D voxels using the latent set bottlenecked self-attention operation of Perceiver [133], whose complexity is linear to the number of voxels as opposed to quadratic, as the all-to-all self attention operations. However, it gives up on spatial disentanglement of features due to the latent set bottleneck. Other methods avoid featurizing points in 3D free space altogether and instead regress an offset for the robot's 3D locations from a detected 2D image contact point [103, 203, 258], which again does not fully comply with spatial equivariance.

In this chapter, we introduce Act3D, a language-conditioned transformer for multi-task 6 DoF robot manipulation that predicts continuous resolution 3D action maps through adaptive 3D spatial computation. Act3D represents the scene as a continuous 3D feature field. It computes a scene-level physical 3D feature cloud by lifting features of 2D foundational models from one or more views using sensed depth. It learns a 3D feature field of arbitrary spatial resolution via recurrent coarse-to-fine 3D point sampling and featurization. At each iteration, the model samples 3D points in the whole workspace and featurizes them using relative spatial cross-attention [325] to the physical 3D feature cloud. Act3D predicts 3D end-effector locations by scoring 3D point features, and then regresses the 3D orientation and opening of the end-effector. At inference time, we can trade-off compute for higher spatial precision and task performance by sampling more 3D points in free space than the model ever saw at training time.

We test Act3D in RLBench [139], an established benchmark for learning diverse robot manipulation policies from demonstrations. We set a new state-of-the-art in the benchmark in both single-task and multi-task settings. Specifically, we achieve a 10% absolute improvement over prior SOTA on the single-task setting introduced by HiveFormer [103] with 74 tasks and a 22% absolute improvement over prior SOTA in the multi-task setting introduced by PerAct [312] with 18 tasks and 249 variations. We also validate our approach on a Franka Panda with a multi-task agent trained from scratch on 8 real-world tasks with a total of just 100 demonstrations (see Figure 6.2). In thorough ablations, we show the importance of the design choices of our architecture, specifically, relative spatial attention, large-scale vision-language pre-trained 2D backbones, high resolution featurization and weight tying across coarse-to-fine attentions.

In summary, our contributions are: **1.** A novel neural policy architecture for language-conditioned multi-task 6-DoF manipulation that both reasons directly in 3D and preserves locality of computation in 3D, using iterative coarse-to-fine translation-invariant attention. **2.** Strong empirical results on a range of simulated and real-world tasks, outperforming the previous SOTA 2D and 3D methods on RLBench by large absolute margins, and generalizing well to novel camera placements at test time. **3.** Thorough ablations that quantify the contribution of high-resolution features, tied attention weights, pre-trained 2D features, and relative position attention design choices. Code and videos are available at our project website: https://act3d.github.io/.

6.2 Related Work

Learning robot manipulation from demonstrations Many recent work train multi-task manipulation policies that leverage Transformer architectures [19, 103, 199, 284, 307, 312] to predict robot actions from video input and language instructions. End-to-end image-to-action policy models, such as RT-1 [19], GATO [284], BC-Z [141], and InstructRL [199], directly predict 6-DoF end-effector poses from 2D video and language inputs. They require many thousands of demonstrations to learn spatial reasoning and generalize to new scene arrangements and environments. Transporter networks [396] and their subsequent variants [93, 304, 311] formulate 4-DoF end-effector pose prediction as pixel classification in 2D overhead images. Thanks to the spatial equivariance of their architecture, their model dramatically increased sample efficiency over previous methods that regress end-effector poses by aggregating global scene features. However, they are limited to top-down 2D planar worlds with simple pick-and-place primitives. 3D policy models of C2F-ARM [140] and PerAct [312] voxelize the robot's workspace and are trained to detect the 3D voxel that contains the next end-effector keypose. Spatially precise 3D pose prediction requires the 3D voxel grid to be high resolution, which comes at a high computational cost. C2F-ARM [140] uses a coarse-to-fine voxelization to handle computational complexity, while PerAct [312] uses Perceiver's latent bottleneck [133] to avoid voxel-to-voxel self-attention operations. Act3D avoids 3D voxelization altogether and instead represents the scene as a continuous resolution 3D feature field. It samples 3D points in the empty workspace and featurizes them using cross-attentions to the physical 3D point features.

Feature pre-training for robot manipulation Many 2D policy architectures bootstrap learning from demonstrations from frozen or finetuned 2D image backbones [141, 249, 259, 382] to increase experience data sample efficiency. Pretrained vision-language backbones can enable generalization to new instructions, objects, and scenes [311, 324]. In contrast, SOTA 3D policy models are typically trained from scratch from colored point clouds input [140, 312, 363]. Act3D uses CLIP pre-trained 2D backbones [276] to featurize 2D image views and lifts the 2D features in 3D using depth [107, 337]. We show that 2D feature pretraining gives a considerable performance boost over training from scratch.

Relative attention layers Relative attentions have shown improved performance in many 2D visual understanding tasks and language tasks [217, 309]. Rotary embeddings [325] implement relative attention efficiently by casting it as an inner-product in an extended position feature space. In 3D, relative attention is imperative as the coordinate system is arbitrary. 3D relative attentions have been used before in 3D Transformer architectures for object detection and point labelling [377, 403]. We show in Section 6.4 that relative attentions significantly boost performance of our model.

6.3 3D Feature Field Transformers for Multi-Task Robot Manipulation

The architecture of Act3D is shown in Figure 6.1. It is a policy transformer that, at a given timestep t, predicts a 6-DoF end-effector pose from one or more RGB-D images, a language instruction, and proprioception information regarding the robot's current end-effector pose. Following prior work [103, 138, 199, 312], instead of predicting an end-effector pose at each timestep, we extract a set of *keyposes* that capture bottleneck end-effector poses in a demonstration. A pose is a keypose if (1) the end-effector changes state (something is grasped or released) or (2) velocities approach near zero (a common occurrence when entering pre-grasp poses or entering a new phase of a task). The prediction problem then boils down to predicting the next (best) keypose action given the current observation. At inference time, Act3D iteratively predicts the next best keypose and reaches it with a sampling-based motion planner, following previous works [103, 312].

We assume access to a dataset of n demonstration trajectories. Each demonstration is a sequence of observations $O = \{o_1, o_2, .., o_t\}$ paired with continuous actions $A = \{a_1, a_2, .., a_t\}$ and, optionally, a language instruction l that describes the task. Each observation o_t consists of RGB-D images from one or more camera views; more details are in Appendix E.2. An action a_t consists of the 3D position and 3D orientation (represented as a quaternion) of the robot's end-effector, its binary open or closed state, and whether the motion planner needs to avoid collisions to reach the pose:

$$a = \{a_{\text{pos}} \in \mathbb{R}^3, a_{\text{rot}} \in \mathbb{H}, a_{\text{open}} \in \{0, 1\}, a_{\text{col}} \in \{0, 1\}\}$$

Next, we describe the model's architecture in detail.

Visual and language encoder Our visual encoder maps multi-view RGB-D images into a multi-scale 3D scene feature cloud. We use a large-scale pre-trained 2D feature extractor followed by a feature pyramid network [195] to extract multi-scale visual tokens for each camera view. Our input is RGB-D, so each pixel is associated with a depth value. We "lift" the extracted 2D feature vectors to 3D using the pinhole camera equation and the camera intrinsics, based on their average depth. The language encoder featurizes instructions with a large-scale pre-trained language encoder. We use the CLIP ResNet50 [276] visual encoder and language encoders to exploit their common vision-language feature space for interpreting instructions and referential grounding. Our pre-trained visual and language encoders are frozen, not finetuned, during training of Act3D.

Iterative 3D point sampling and featurization Our key idea is to estimate high resolution 3D action maps by learning 3D perceptual representations of free space with arbitrary spatial resolution, via recurrent coarse-to-fine 3D point sampling and featurization. 3D point candidates (which we will call ghost points) are sampled, featurized and scored iteratively through relative cross-attention [325] to the physical 3D scene feature cloud, lifted from 2D feature maps of the input image views. We first sample coarsely across the entire workspace, then finely in the vicinity of the ghost point selected as the focus of attention in the previous iteration, as shown in Figure 6.1. The coarsest ghost points attend to a global coarse scene feature cloud, whereas finer ghost points attend to a local fine scene feature cloud.

Relative 3D cross-attentions We featurize each of the 3D ghost points and a parametric query (used to select via inner-product one of the ghost points as the next best end-effector position in the decoder) independently through cross-attentions to the multi-scale 3D scene feature cloud, language tokens, and proprioception. Featurizing ghost points independently, without self-attentions to one another, enables sampling more ghost points at inference time to improve performance, as we show in Section 6.4. Our cross-attentions use relative 3D position information and are implemented efficiently with rotary positional embeddings [325]. The absolute locations of our 3D points are never used in our featurization, and attentions only depend on the relative locations of two features.

Decoding actions We score ghost point tokens via inner product with the parametric query to select one as the next best end-effector position a_{pos}. We then regress the end-effector orientation a_{rot} and opening a_{open}, as well as whether the motion planner needs to avoid collisions to reach the pose a_{col}, from the last iteration parametric query with a 2-layer multi-layer perceptron (MLP).

Training Act3D is trained supervised from input-action tuples from a dataset of manipulation demonstrations. These tuples are composed of RGB-D observations, language goals, and keypose actions $\{(o_1, l_1, k_1), (o_2, l_2, k_2), ...\}$. During training, we randomly sample a tuple and supervise Act3D to predict the keypose action k given the observation and goal (o, l). We supervise position prediction a_{pos} at every round of coarse-to-fine with a softmax cross-entropy loss over ghost points, rotation prediction a_{rot} with a MSE loss on the quaternion prediction, and binary end-effector opening a_{open} and whether the planner needs to avoid collisions a_{col} with binary cross-entropy losses.

Implementation details We use three ghost point sampling stages: first uniformly across the entire workspace (roughly 1 meter cube), then uniformly in a 16 centimeter diameter ball, and finally in a 4 centimeter diameter ball. The coarsest ghost points attend to a global coarse scene feature cloud (32x32xn_{cam} coarse visual tokens) whereas finer ghost points attend to a local fine scene feature cloud (the closest 32x32xn_{cam} out of the total 128x128xn_{cam} fine visual tokens). During training, we sample 1000 ghost points in total split equally across the three stages. At inference time, we can trade-off extra prediction precision and task performance for additional compute by sampling more ghost points than the model ever saw at training time ($10,000$ in our experiments). We'll show in ablations in Section 6.4 that our framework is robust to these hyper-parameters but tying weights across sampling stages and relative 3D cross-attention are both crucial for generalization. We use a batch size 16 on a Nvidia 32GB V100 GPU for 200k steps (one day) for single-task experiments, and a batch size 48 on 8 Nvidia 32GB V100 GPUs for 600K steps (5 days) for language-conditioned multi-task experiments. At test time, we call upon a low-level motion planner to reach predicted keyposes. In simulation, we use native motion planner implementation provided in RLBench, which is a sampling-based BiRRT [176] motion planner powered by Open Motion Planning Library (OMPL) [327] under the hood. For real-world experiments, we use the same BiRRT planner provided by the MoveIt! ROS package [44]. please, see Appendix E.4 for more details.

6.4 Experiments

We test Act3D in learning from demonstrations single-task and multi-task manipulation policies in simulation and the real world. We conduct our simulated experiments in RLBench [139], an established simulation benchmark for learning manipulation policies, for the sake of reproducibility and benchmarking. Our experiments aim to answer the following questions:

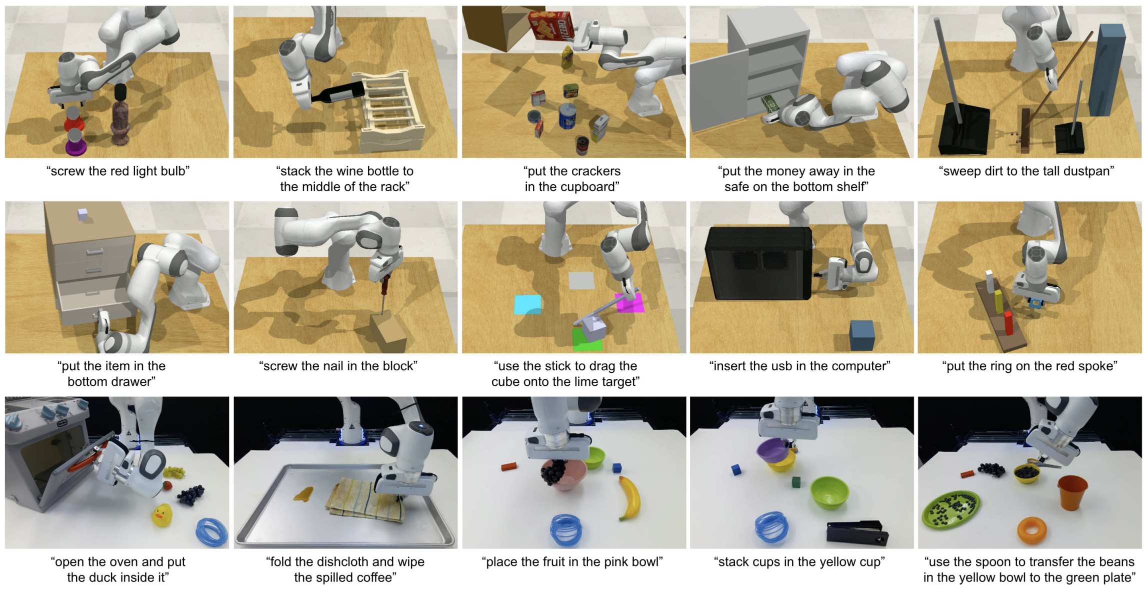

"screw the red light bulb"
"stack the wine bottle to the middle of the rack"
"put the crackers in the cupboard"
"put the money away in the safe on the bottom shelf"
"sweep dirt to the tall dustpan"
"put the item in the bottom drawer"
"screw the nail in the block"
"use the stick to drag the cube onto the lime target"
"insert the usb in the computer"
"put the ring on the red spoke"
"open the oven and put the duck inside it"
"fold the dishcloth and wipe the spilled coffee"
"place the fruit in the pink bowl"
"stack cups in the yellow cup"
"use the spoon to transfer the beans in the yellow bowl to the green plate"

Figure 6.2: **Tasks.** We conduct experiments on 92 simulated tasks in RLBench [139] (only 10 shown), and 8 real-world tasks (only 5 shown).

1. How does Act3D compare against SOTA 2D multiview and 3D manipulation policies in single-task and multi-task settings with varying number of training demonstrations?

2. How does Act3D generalize across camera viewpoints compared to prior 2D multiview policies?

3. How do design choices such as relative 3D attention, pre-trained 2D backbones, weight-tied attention layers, and the number of coarse-to-fine sampling stages impact performance?

6.4.1 Evaluation in simulation

Datasets We test Act3D in RLbench in two settings: **1. Single-task** manipulation policy learning. We consider 74 tasks grouped into 9 categories proposed by HiveFormer [103]. Each task includes variations which test generalization to novel arrangements of the same training objects. Each method is trained with 100 demonstrations and evaluated on 500 unseen episodes. **2. Multi-task** manipulation policy learning. We consider 18 tasks with 249 variations proposed by PerAct [312]. Each task includes 2-60 variations, which test generalization to new goal configura-

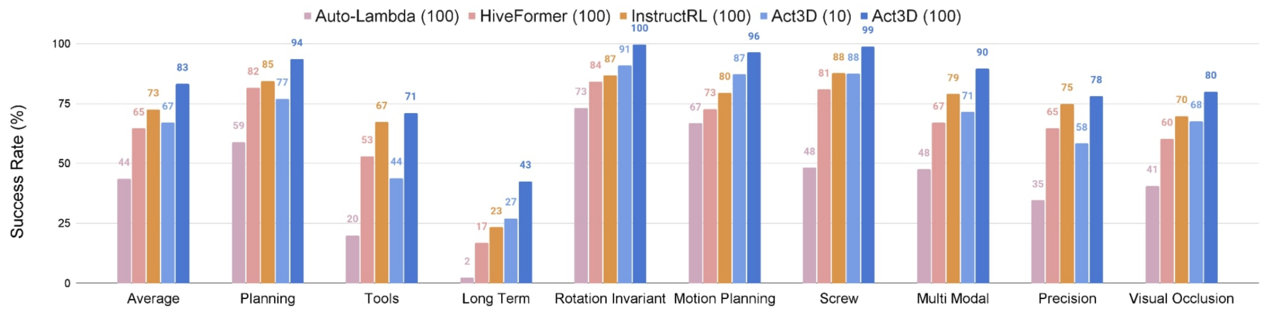
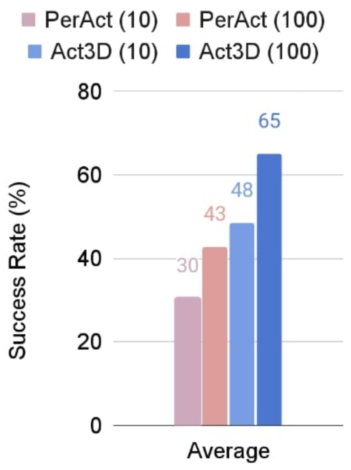


Figure 6.3: **Single-task performance.** On 74 RLBench tasks across 9 categories, Act3D reaches 83% success rate, an absolute improvement of 10% over InstructRL [199], prior SOTA in this setting.



	open drawer		slide block		sweep to dustpan		meat off grill		turn tap		put in drawer		close jar		drag stick		stack blocks	
Method	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100
PerAct	68	80	32	72	72	56	**68**	84	**72**	80	16	68	32	60	36	68	**12**	**36**
Act3D	**92**	**93**	**66**	**93**	**82**	**92**	58	**94**	64	**94**	**82**	**90**	**90**	**92**	**52**	**92**	6	12
	screw bulb		put in safe		place wine		put in cupboard		sort shape		push buttons		insert peg		stack cups		place cups	
	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100	10	100
PerAct	**28**	24	16	44	20	12	0	16	**16**	**20**	56	48	4	0	0	0	0	0
Act3D	26	**47**	**75**	**95**	**32**	**80**	**27**	**51**	7	8	**98**	**99**	**7**	**27**	**8**	**9**	**1**	**3**

Figure 6.4: **Multi-task performance.** On 18 RLBench tasks with 249 variations, Act3D reaches 65% success rate, an absolute improvement of 22% over PerAct [312], prior SOTA in this setting.

tions that involve novel object colors, shapes, sizes, and categories. This is a more challenging setting. Each method is trained with 100 demonstrations per task split across variations, and evaluated on 500 unseen episodes per task.

Baselines We compare Act3D with the following state-of-the-art manipulation policy learning methods: **1.** InstructRL [199], a 2D policy that directly predicts 6 DoF poses from image and language conditioning with a pre-trained vision-and-language backbone. **2.** PerAct [312], a 3D policy that voxelizes the workspace and detects the next best voxel action through global self-attention. **3.** HiveFormer [103] and

Auto-λ [203], hybrid methods that detect a contact point within an image input, then regress an offset from this contact point. We report numbers from the papers when available.

Evaluation metric We evaluate policies by task completion success rate, the proportion of execution trajectories that lead to goal conditions specified in language instructions.

Single-task and multi-task manipulation results We show single-task quantitative results of our model and baselines in Figure 6.3. Act3D **reaches 83% success rate, an absolute improvement of 10% over InstructRL [199], prior SOTA in this setting**, and consistently outperforms it across all 9 categories of tasks. With only 10 demonstrations per task, Act3D is competitive with prior SOTA using 100 demonstrations per task. Act3D outperforms 2D methods of InstructRL and Hiveformer because it reasons directly in 3D. For the same reason, it generalizes much better than them to novel camera placements, as we show in Table 6.1.

We show multi-task quantitative results of our model and PerAct in Figure 6.4. Act3D reaches 65% success rate, an absolute improvement of 22% over PerAct, prior SOTA in this setting, consistently outperforming it across most tasks. **With only 10 demonstrations per task, Act3D outperforms PerAct using 100 demonstrations per task.** Note that Act3D also uses less than a third of PerAct's training computation budget: PerAct was trained for 16 days on 8 Nvidia V100 GPUs while we train for 5 days on the same hardware. Act3D outperforms PerAct because its coarse-to-fine relative attention based 3D featurization of the 3D workspace is more effective than the perceiver's latent bottleneck attention in generating spatially disentangled features.

6.4.2 Ablations

We ablate the impact of our design choices in Table 6.1. We perform most ablations in the single-task setting on 5 tasks: pick cup, put knife on chopping board, put money in safe, slide block to target, take umbrella out of stand. We ablate the choice of pre-trained 2D backbone in the multi-task setting with all 18 tasks.

Table 6.1: **Ablations.**

		Average success rate in single-task setting (5 tasks)
Core design choices	Full Act3D	**98.1**
	Only 2 stages of coarse-to-fine sampling	93.6
	No weight tying across stages	80.6
	Absolute 3D positional embeddings	55.4
	Attention to only global coarse visual features	89.8
	Only 1000 ghost points at inference time	93.2
Viewpoint changes	Act3D	**74.2**
	HiveFormer	20.4
		Multi-task setting (18 tasks)
Backbone	CLIP ResNet50 backbone	**65.1**
	ImageNet ResNet50 backbone	53.4

Generalization across camera viewpoints: We vary camera viewpoints at test time for both Act3D and HiveFormer [103]. The success rate drops to 20.4% for HiveFormer, a relative 77% drop, while Act3D achieves 74.2% success rate, a 24% relative drop. This shows detecting actions in 3D makes Act3D more robust to camera viewpoint changes than multiview 2D methods that regress offsets.

Weight-tying and coarse-to-fine sampling: All 3 stages of coarse-to-fine sampling are necessary: a model with only 2 stages of sampling and regressing an offset from the position selected at the second stage suffers a 4.5% performance drop. Tying weights across stages and relative 3D positional embeddings are both crucial; we observed severe overfitting without, reflected in respective 17.5% and 42.7% performance drops. Fine ghost point sampling stages should attend to local fine visual features with precise positions: all stages attending to global coarse features leads to a 8.3% performance drop. Act3D can effectively trade off inference computation for performance: sampling 10,000 ghost points, instead of the 1,000 the model was trained with, boosts performance by 4.9%.

Pre-training 2D features: We investigate the effect of the pre-trained 2D backbone in the multi-task setting where language instructions are most needed. A ResNet50 [276] backbone pre-trained with CLIP improves success rate by 8.7% over a ResNet50 backbone pre-trained on ImageNet.

We found Random crops of RGB-D images to boost performance but yaw rotation perturbations did not help. The model is robust to variations in hyperparameters such as the diameter of ghost point sampling balls or the number of points sampled during training. For additional ablations regarding augmentations and sensitivity to hyperparameters, please see the Appendix section E.6.

6.4.3 Evaluation in real-world

In our real-world setup, we conduct experiments with a Franka Emika Panda robot and a single Azure Kinect RGB-D sensor. We consider 8 tasks (Figure 6.2) that involve interactions with multiple types of objects, spanning liquid, articulated objects, and deformable objects. For each task, we collected 10 to 15 kinesthetic demonstrations and trained a languaged-conditioned multi-task model with all of them. We report the success rate on 10 episodes per task in Table 6.2. Act3D can capture semantic knowledge in demonstration well and performs reasonably well on all tasks, even with a single camera input. One major failure case comes from noisy depth sensing: when the depth image is not accurate, the selected point results in imprecise action prediction. Leveraging multi-view input for error correction could improve this, and we leave this for future work. For videos of the robot executing the tasks, please see our project website.

Task	# Train	Success
reach target	10	10/10
duck in oven	15	6/10
wipe coffee	15	7/10
fruits in bowl	10	8/10
stack cups	15	6/10
transfer beans	15	5/10
press handsan	10	10/10
uncrew cap	10	8/10

Table 6.2: Real-world tasks.

6.4.4 Limitations and future work

Our framework currently has the following limitations: **1.** Act3D is limited by the motion planner used to connect predicted keyposes with straight trajectory segments. It does not handle manipulation of articulated object well, such as opening/closing doors, fridges, and ovens, where robot trajectories cannot be well approximated by few line segments.**2.** Act3D does not utilize any decomposition of tasks into subtasks. A hierarchical framework that would predict language subgoals for subtasks [3, 128, 191] and feed those to our language-conditioned policy would allow better re-usability of skills across tasks. Addressing these limitations is a direct avenue for future work.

6.5 Conclusion

We presented Act3D, a language-conditioned policy transformer that predicts continuous resolution 3D action maps for multi-task robot manipulation. Act3D represents the scene using a continuous resolution 3D feature map, obtained by coarse-to-fine 3D point sampling and attention-based featurization. Act3D sets a new state-of-the-art in RLBench, an established robot manipulation benchmark, and solves diverse manipulation tasks in the real world from a single RGB-D camera view and a handful of demonstrations. Our ablations quantified the contribution of relative 3D attentions, 2D feature pre-training, and weight tying during coarse-to-fine iterations.

Chapter 7

ChainedDiffuser: Unifying Trajectory Diffusion and Keypose Prediction for Robotic Manipulation

An obvious extension of Act3D, this chapter tackles the non-learnable planner limitation by introducing our first diffusion-based formulation.

7.1 Introduction

While learning manipulation policies from demonstrations is a supervised learning problem, the multimodality and diversity of action trajectories poses significant challenges to machine learning methods. Some tasks, such as placing a cup in a cabinet, can be handled by a policy that provides only a desired goal pose for the cup [311, 312, 395], while others, such as wiping off dirt on the floor, necessitate the policy to generate a continuous action trajectory [229, 339] for the grasped mop.

One line of manipulation learning methods models action trajectories from demonstrations. These methods either reactively map vision and language to dense temporal

This chapter is based on the paper previously published at CoRL 2023 [364]

actions [19, 229, 307, 315, 402], or model the input-action compatibility using energy-based models [69, 82, 93, 145]. Despite recent progress, these methods may struggle with multimodal action trajectory distributions, or experience training stabilities [38, 69, 143]. Building on successes in diffusion models [118, 252, 321], a recent line of work proposes to train diffusion-based policies [38, 286, 339] for generating action trajectories. These approaches have demonstrated stable training behavior and impressive capability in capturing multimodal action trajectory distributions. Yet, they have not yet been tested on long-horizon manipulation tasks.

Another line of works casts the problem of robot manipulation as predicting a sequence of discrete end-effector actions on keyframes [91, 93, 304, 395]. This paradigm extracts keyframes from continuous demonstrations and predicts end-effector actions in these keyframes [91, 199, 311, 312]. Subsequently, a low-level path planner connects the predicted keyposes (*macro-actions*), and returns full trajectories that adhere to both environmental and task constraints. Leveraging recent advances in attention-based architectures [342], a number of methods extend keyframe action prediction to 6-DoF language-instructed manipulation tasks [91, 103, 199, 311, 312].

The assumptions behind keyframe prediction hinder its applicability to manipulation tasks that extend beyond pick-and-place type of actions. Many tasks, such as wiping a table, opening a door while respecting the kinematic constraints, etc., can only be solved via continuous interactions with the environment. Moreover, the dependence on low-level path planning further restricts these methods' capability: while a range of tasks need collision-free trajectories, other tasks, such as object pushing [103, 312, 338], necessitate that the motion planner disregards collision avoidance. Although supervision for this additional reasoning is readily available in simulated datasets [139], real-world human demonstrations typically lack such data, not to mention that collision-free motion planning in the real world requires accurate state estimation, which presents its own challenges.

In light of the above, we present *ChainedDiffuser*, a neural architecture that unifies the two aforementioned paradigms. ChainedDiffuser is a policy architecture that takes as input visual signals and, optionally, a language instruction and outputs temporally dense end-effector actions. At a coarse level, it predicts macro-step end-effector actions (which we will call *macro-actions*), a high-level task that requires global comprehension of the visual environment and the task to complete, with a global transformer-based action predictor. Then, a low-level trajectory diffuser generates local trajectory segments to connect the predicted macro-actions. In comparison to transformer-based macro-step prediction methods [103, 199, 311, 312], our model predicts smooth trajectories to accommodate tasks that require continuous interactions and collision-free actions. In comparison to diffusion-only trajectory generation methods [38, 143, 286, 339], our hierarchical approach handles long-horizon tasks in a more structured manner and allows different modules to concentrate on the tasks at which they excel.

We test ChainedDiffuser on RLBench [139], an established benchmark for manipulation learning from demonstrations. We evaluate our model across a variety of tasks and scenarios studied in previous literature [103, 199]. ChainedDiffuser sets a new state of the art, and outperforms ablative versions that do not predict macro-actions or use regression or motion planners for keyframe-to-keyframe trajectory prediction. Furthermore, we validate our model in real-world scenarios with a number of long-horizon manipulation tasks, using a handful of human demonstrations for training. Code and videos are available at our project website: https://chained-diffuser.github.io/.

7.2 Related Work

Learning from Demonstrations [10, 267] has been a common paradigm for robotics but requires demonstration data collection in the real world [19, 324, 389] or simulation [139, 150, 246]. To improve data efficiency, several approaches learn the policy on top of pre-trained visual representations that exploit large vision-only datasets [249, 259, 277, 305, 306, 367]. Orthogonal to this, other approaches abstract every task as a sequence of subgoals, expressed as pick-and-place primitives [311, 395] or keyframes [138, 140]. In this case, hand-designed low-level controllers are employed to plan the end-effector's motion between intermediate subgoals. While data-efficient, this abstraction does not generalize adequately to scenarios where only few specific trajectories that respect all physical constraints are valid [140], such as manipulations of deformable [365, 373] or articulated [89] objects, motions of closed-chain robotic systems [326, 362], or trajectories through obstacles in a cluttered environment [80]. As a result, recent works resort to semi-manual cost specification for each additional constraint (e.g., collision avoidance, trajectory smoothness [339]). Closer to our approach, James et al. [140] learn to score trajectories proposed by either hand-designed or learning-based planners. Instead, we train scene conditioned diffusion models to generate trajectories that connect predicted keyposes.

Transformers for Robotics Following their success in natural language processing [22, 57, 342] and computer vision [65, 115], numerous recent works use Transformer-based architectures for robotics and control [19, 27, 91, 142, 206, 278]. One main motivation is the flexibility of attention for long-horizon prediction when combining information from multiple sensory streams, such as visual observations and language instructions [66, 199]. Most related to ours is the stream of multi-tasking Transformer-based models, that are trained on diverse datasets to achieve higher in-distribution [103, 312] or out-of-distribution generalization [19, 180, 371, 372]. Our model comprises of two attention-based modules, one for macro-step action prediction and one for local trajectory optimization, that can leverage different input modalities and operate over different abstractions.

Diffusion Models [118, 252, 320, 321] learn to approximate the data distribution through an iterative denoising process, and have shown impressive results on both unconditional and conditional image generation [58, 280, 292, 298]. In the field of robotics, diffusion models find applications on planning [4, 125, 143], scene re-arrangement [159, 208], controllable motion optimization [147, 409], video generation [71] and imitation learning [38, 286]. Their main advantage is that they can better capture the action trajectory distribution compared to previous generative models. Recent works use diffusion model to predict complete trajectories, often auto-regressively [38, 261]. Instead, we use diffusion models to generate local trajectories that are chained together with macro-actions.

7.3 ChainedDiffuser

7.3.1 Overview

The architecture of ChainedDiffuser is illustrated in Figure 7.1. ChainedDiffuser combines macro-action prediction with conditional trajectory diffusion. Its input comprises of visual observations of the environment and a natural language description

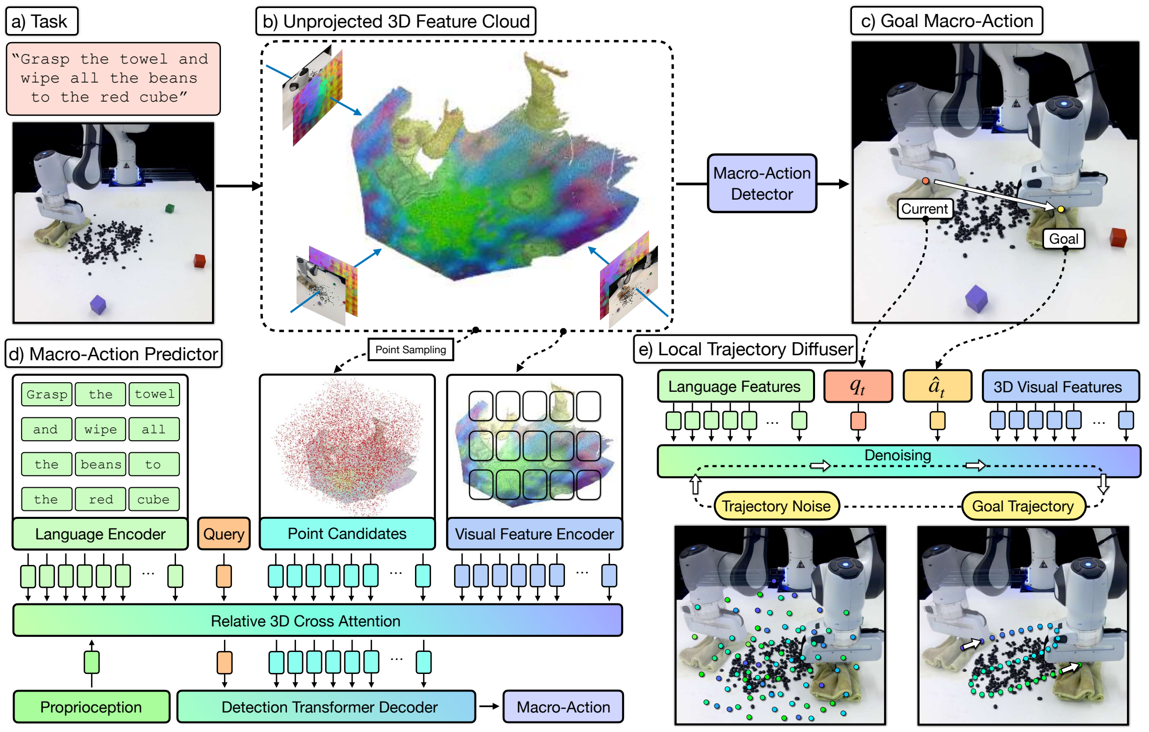

a) Task
"Grasp the towel and wipe all the beans to the red cube"
b) Unprojected 3D Feature Cloud
c) Goal Macro-Action
Macro-Action Detector
Current
Goal
d) Macro-Action Predictor
Point Sampling
e) Local Trajectory Diffuser
Grasp the towel and wipe all the beans to the red cube
Language Encoder
Query
Point Candidates
Visual Feature Encoder
Relative 3D Cross Attention
Proprioception
Detection Transformer Decoder
Macro-Action
Language Features
q_t
$\hat{a}_t$
3D Visual Features
Denoising
Trajectory Noise
Goal Trajectory

Figure 7.1: **ChainedDiffuser** is a robot manipulation policy architecture that predicts a set of robot keyposes and links them using predicted trajectory segments. It featurizes input multi-view images using pre-trained 2D image backbones and lifts the resulting 2D feature maps to 3D using sensed depth. In (b) we visualize the 3D feature cloud using PCA and keeping the 3 principal components, mapping them to RGB. The model then predicts end-effector keyposes using coarse-to-fine attention operations to estimate a 3D action map for the end-effector's 3D location and regress the robot's 3D orientation, similar to [91] (d). It then links the current end-effector pose to the predicted one with a trajectory predicted using a diffusion model conditioned on the 3D scene feature cloud and predicted keypose (e).

l of the task. At each step, ChainedDiffuser predicts a macro-action $\hat{a}_t$ using a global policy π_{global}, and then feeds $\hat{a}_t$ together with its current end-effector state q_t to a low-level local trajectory generator $\pi_{\text{local}}(q_t, \hat{a}_t)$ to generate dense micro-actions connecting q_t and $\hat{a}_t$, as shown in Figure 7.1. Both a_t and q_t share the same space $\mathcal{A} = \{a_{\text{pos}}, a_{\text{rot}}, a_{\text{grip}}\}$, consisting of the end-effector's 3D position a_{pos}, rotation a_{rot} represented as a 4D quaternion, and a binary flag a_{grip} indicating whether the gripper is open. For each task, we assume access to a dataset $\mathcal{D} = \{\zeta_1, \zeta_2, ..., \zeta_m\}$ of m expert demonstrations, where ζ_i contains the language instructions l, visual observations o

and end-effector states q_t for all timesteps in the demonstration.

Input Encoding ChainedDiffuser operates in a 3D space to achieve robustness across changing camera viewpoints – an important advantage over prior 2D methods which assume fixed camera viewpoints [38, 103, 199]. Compared to prior robotic architectures which rely on voxel-based 3D representation (e.g., [140, 312]), ChainedDiffuser employs a point-based representation, that facilitates sparse computation and circumvents precision loss during voxelization. ChainedDiffuser uses a frozen CLIP [276] to encode both the language instruction l and the RGB images o_t into a set of language and visual feature tokens respectively. Then, it uses the depth channel information to unproject the 2D image feature tokens into a 3D feature cloud (Figure 7.1(b)), where each visual token has 2D appearance information and 3D positional information. We also encode the proprioception information q_t with a simple MLP.

7.3.2 Macro-Action Predictor

Our macro-action predictor π_{global} is based on Act3D [91], a state-of-the-art macro-action prediction method that uses a point-based transformer that casts end-effector action prediction as 3D action map prediction. We include its main pipeline here for completeness. Act3D samples iteratively 3D point candidates and featurizes them using relative position attentions to a scene 3D feature cloud. Then, a trainable query token $\mathcal{Z}_{\text{query}}$ is used to score a pool of N point candidates $\{P_i = \langle x_i, y_i, z_i \rangle\}_{i=1}^{N}$ in the scene and select a position for next macro-action. The point candidates are first uniformly sampled within the robot's empty workspace and only contain 3D positional information and a trainable feature embedding $\mathcal{Z}_{\text{point}}$. The query token and the point candidates individually attend to the concatenation (across the sequence dimension) of language tokens $\mathcal{Z}_{\text{ins}}$, visual feature tokens $\mathcal{Z}_{\text{vis}}$ and proprioception token $\mathcal{Z}_{\text{robot}}$ (Figure 7.1(d)):

$$\tilde{\mathcal{Z}}_{\text{query}} = \text{Attn}\Big(\mathcal{Z}_{\text{query}}, \langle \mathcal{Z}_{\text{ins}}, \mathcal{Z}_{\text{vis}}, \mathcal{Z}_{\text{robot}} \rangle\Big) \tag{7.1}$$

$$\tilde{\mathcal{Z}}_{\text{point}} = \text{Attn}\Big(\mathcal{Z}_{\text{point}}, \langle \mathcal{Z}_{\text{ins}}, \mathcal{Z}_{\text{vis}}, \mathcal{Z}_{\text{robot}} \rangle\Big) \tag{7.2}$$

where $\mathrm{Attn}(x, y)$ is an attention operation [222, 342] where the queries are formed from x, the keys and values from y. After this contextualization step, the query token and the point candidates have captured the task and scene information. We take the dot product of the contextualized query embedding with all point candidates and select the best-matching point candidate for the position of the predicted macro-action:

$$\hat{a}_{\text{pos}} = \langle x_{\hat{i}}, y_{\hat{i}}, z_{\hat{i}} \rangle, \quad \hat{i} = \underset{i}{\operatorname{argmax}}\, \tilde{\mathcal{Z}}_{\text{query}}^{T} \cdot \tilde{\mathcal{Z}}_{\text{point}}^{i} \tag{7.3}$$

Once we obtain the best point candidate, we predict the rotation and gripper open flag with a simple MLP on top of the query:

$$\langle \hat{a}_{\text{rot}}, \hat{a}_{\text{grip}} \rangle = \mathrm{MLP}(\tilde{\mathcal{Z}}_{\text{query}}) \tag{7.4}$$

7.3.3 Local Trajectory Diffuser

Once we obtain the macro-action $\hat{a}_t$ for the current step t, we call upon our diffusion-based local trajectory generator to fill up the gap in-between with micro-actions. We model such trajectory generation as a denoising process [38, 118, 339]: we start with drawing a sequence of S random Gaussian samples $\{\mathbf{x}_s^K\}_{s=1}^S$ in the normalized SE(3) space, and then perform K denoising iterations to transform the noisy trajectories to a sequence of noise-free waypoints $\{\mathbf{x}_s^0\}_{s=1}^S$. Each denoising iteration is described by:

$$\mathbf{x}_s^{k-1} = \lambda_k(\mathbf{x}_s^k - \gamma_k \epsilon_\theta(\mathbf{x}_s^k, k)) + \mathcal{N}(0, \sigma_k^2 I), \quad 1 \leq s \leq S \tag{7.5}$$

where ϵ_θ is the noise prediction network, k the denoising step, $\mathcal{N}(0, \sigma_k^2 I)$ the Gaussian noise added at each iteration, and $\lambda_k, \gamma_k, \sigma_k$ are scalar noise schedule functions dependent on k (Appendix F.1).

The noise prediction network (Figure 7.1 (e)) is also an attention-based model that absorbs similar input as the macro-action selector does, i.e., the language instruction l, RGB-D observations o_t and current end-effector state q_t, but additionally conditions on the goal macro-action $\hat{a}_t$ and the denoising timestep k. The language tokens $\mathcal{Z}_{\text{ins}}$, visual tokens $\mathcal{Z}_{\text{vis}}$ and current end-effector state $\mathcal{Z}_{\text{robot}}$ are featurized similarly to the Macro-Action Selector. We use an MLP to encode the goal macro-action into $\mathcal{Z}_{\text{macro}} = \mathrm{MLP}(\hat{a}_t)$. We encode the denoising timestep into $\mathcal{Z}_{\text{time}}$ using sinusoidal positional embeddings [342], and encode the the sampled noise using an MLP into a sequence of tokens $\mathcal{Z}_s^k$. We let this sequence iteratively cross-attend to all encoded inputs first:

$$\tilde{\mathcal{Z}}_s^k = \text{Attn}(\mathcal{Z}_s^k, \langle \mathcal{Z}_{\text{ins}}, \mathcal{Z}_{\text{vis}}, \mathcal{Z}_{\text{robot}}, \mathcal{Z}_{\text{macro}}, \mathcal{Z}_{\text{time}} \rangle),$$

and then self-attend to obtain a finalized $\tilde{\mathcal{Z}}_s^k$ (note that we reuse the same symbol for presentation clarity):

$$\tilde{\mathcal{Z}}_s^k = \text{Attn}(\tilde{\mathcal{Z}}_s^k, \tilde{\mathcal{Z}}_s^k)$$

Again, we use relative positional embeddings to encode all tokens' spatial positions. For the trajectory noise tokens, we additionally encode each sample's temporal position s using sinusoidal positional embeddings. These are added to the respective noise tokens $\mathcal{Z}_s^k$. The contextualized noise sample is then fed into another MLP for noise regression:

$$\epsilon_\theta(\mathbf{x}_s^k, k) = \text{MLP}(\tilde{\mathcal{Z}}_s^k) \tag{7.6}$$

After K denoising steps by substituting Equation 7.6 into 7.5, we convert the denoised samples back to the actual micro-actions by unnormalizing them: $a_{t-1+s} = \text{Unnormalize}(\mathbf{x}_s^0), \quad 1 \leq s \leq S$. For more implementation and training details, please see the Appendix 7.3.4.

Noise schedulers We model local trajectory optimization as a discrete-time diffusion process, which we implement using the DDPM sampler [118]. DDPM uses a non-parametric time-dependent noise variance scheduler β_k, which defines how much noise is added at each time step. We adopt a scaled linear schedule for the position and a squared cosine schedule for the rotation of each trajectory step.

7.3.4 Implementation and Training Details

ChainedDiffuser takes as input m multi-view RGB-D images of the scene. For experiments in simulation, we use $m = 3$ (*left*, *right*, *wrist*) or $m = 4$ (with an additional *front* view), depending on the settings of the baselines we compare with. For real-world experiments, we use $k = 1$, with a single front-view camera. Each RGB-D image is 256×256 and is encoded to 64×64 visual tokens with CLIP's ResNet50 visual encoder [276]. The demonstration data contains end-effector states for all timesteps. In order to extract macro-actions to supervise the action selection transformer, we use a simple heuristic following previous literature [103, 199, 312]: a timestep is considered to be a keyframe containing macro-action if the gripper opens or closes, or if the robot arm is not moving (when all joint velocities approach zero). All dense actions present in the demonstration are used to supervise the local trajectory diffuser. We resample the dense trajectories between extracted macro-actions to a trajectory of fixed length $S = 50$. We found in practice, denoising fixed number of micro-actions leads to more stable training, and works better than learning variable-length trajectory diffusion with predicted trajectory length. We train both the action detector and the trajectory diffuser jointly, using a cross-entropy (CE) loss to supervise the point candidate selection by predicting a probability distribution q over all point candidates in the pool, and mean-sqaured error (MSE) losses to supervise quaternion, gripper opening and trajectory noise regression:

$$\mathcal{L} = \frac{1}{|\mathcal{D}||\zeta|} \sum_{\zeta \in \mathcal{D}} \sum_{\hat{t} \in \zeta} \left[\mathrm{CE}(q(\{P_i\}^N), q^*(\{P_i\}^N)) + \mathrm{MSE}(\hat{a}_{\hat{t}}, \hat{a}^*_{\hat{t}}) + \sum_{t=\hat{t}}^{\hat{t}+S-1} \mathrm{MSE}(\epsilon_\theta(\mathbf{x}^k_s, k), \epsilon_k) \right], \tag{7.7}$$

where * indicates predicted value, $\hat{a}_{\hat{t}} = \langle \hat{a}_{\mathrm{rot}}, \hat{a}_{\mathrm{grip}} \rangle$, k is a randomly sampled denoising step, and ϵ_k is the sampled ground truth noise. In order to speed up training in practice, we train the first 2 terms till convergence, and then add the 3rd term for joint optimization, as opposed to using ground-truth macro-actions for training the trajectory diffuser. This allows the trajectory diffuser to incorporate certain error recovery capability to handle inaccurate macro-action predictions. In addition, at test time, we normalize the predicted quaternion to ensure it respects the normalization constraint before feeding it to the robot. We use a batch size of $B = 24$ and AdamW [219] optimizer with a learning rate of $1e - 4$ for all our experiments. Our single-task model is trained for 1 day on one A100 GPU, and multi-task model is trained for 5 days on 4 A100 GPUs.

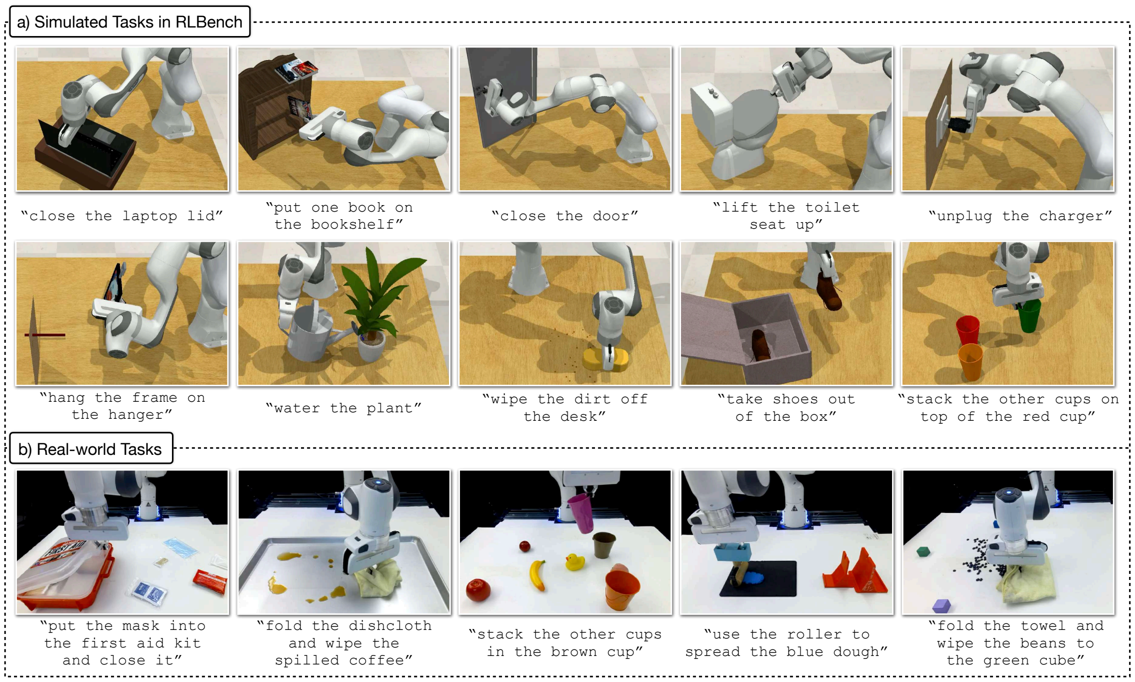

a) Simulated Tasks in RLBench
"close the laptop lid"
"put one book on the bookshelf"
"close the door"
"lift the toilet seat up"
"unplug the charger"
"hang the frame on the hanger"
"water the plant"
"wipe the dirt off the desk"
"take shoes out of the box"
"stack the other cups on top of the red cup"
b) Real-world Tasks
"put the mask into the first aid kit and close it"
"fold the dishcloth and wipe the spilled coffee"
"stack the other cups in the brown cup"
"use the roller to spread the blue dough"
"fold the towel and wipe the beans to the green cube"

Figure 7.2: Simulation and real-world tasks we evaluate on.

Control Our control algorithm is closed-loop at the macro-action level, which means the macro-action predictor reasons about the surroundings and predicts actions to handle environment changes. At the low-level, our controller is open-loop and follows a Cartesian-space end-effector trajectory composed of the predicted micro-actions using position control, with a control frequency of 10Hz. For our real-robot setup, we use the open source `frankapy` package [399] with ROS, which uses a low-level PID controller at 1kHz.

7.4 Experiments

We test ChainedDiffuser in various manipulation tasks in both simulated and real-world environments. Our experiments aim to answer the following questions:

- How does ChainedDiffuser compare to previous SOTA 2D and 3D manipulation methods?
- Is macro-action prediction helpful in guiding trajectory generation?

Table 7.1: Success rates in 10 single-tasks of the Hiveformer experimental setting.

	pick & lift	pick-up cup	push button	put knife	put money	reach target	slide block	stack wine	take money	take umbrella	Mean
Auto-λ [203]	87	78	95	31	62	100	36	23	38	37	55.0
HiveFormer [103]	92	77	100	70	96	100	95	82	82	90	88.4
InstructRL [199]	98	85	**100**	85	**99**	100	**98**	**93**	90	93	93.8
ChainedDiffuser (ours)	**98**	**94**	96	**91**	98	**100**	95	90	**100**	**96**	**95.8**

- Does ChainedDiffuser work in the real-world where only a single camera and limited number of demonstrations are available?

7.4.1 Simulation Experiments

We run experiments in simulation using RLBench [139], a widely adopted manipulation benchmark with diverse tasks concerning interactions with a wide range of objects, as shown in Figure 7.2. We follow the same setting used in prior works [103, 199], where each task has multiple variations and contains 100 demonstrations. We report success rates in each task averaged over 100 unseen test episodes. For baselines, when possible, we use the official numbers reported in their papers.

Baselines We compare ChainedDiffuser with the following baselines:

1. Auto-λ [203] and HiveFormer [103], policy learners that operate on multi-view 2.5D images and predict actions by offseting detected points in the input images.
2. *InstructRL* [199], a policy that operates on multi-view 2D images with pre-trained vision and language encoders, and directly predicts 6-DoF end-effector actions.
3. *Act3D* [91], a policy that predicts keyframe end-effector macro-actions with a 3D action detection transformer and relies on low-level motion planner to connect macro-actions.
4. *Open-loop trajectory diffusion*, which is ChainedDiffuser without the macro-action detector, making it a trajectory diffusion model.
5. *Act3D+ trajectory regression*, which replaces the local trajectory diffuser in ChainedDiffuser with a deterministic trajectory regression

Table 7.2: Success rates on challenging tasks for motion planners.

	unplug charger	close door	open box	open fridge	frame off hanger	open oven	books on shelf	wipe desk	cup in cabinet	shoes out of box	Mean
Act3D [91]	48	9	9	19	66	2	34	4	0	19	21.0
Open-loop trajectory diffusion	65	21	46	37	43	16	40	34	6	9	31.7
Act3D + trajectory regression	95	5	95	60	77	17	68	**70**	40	67	59.6
ChainedDiffuser (ours)	**95**	**76**	**96**	**68**	**85**	**86**	**92**	65	**68**	**78**	**80.9**

Dataset We consider the following single-task experimental settings:

- 10 tasks considered in the Auto-λ [203] experimental setup. These tasks are considered by many prior works and this allows us to compare our performance with them.
- 10 tasks in RLbench we identify to require continuous interaction with the environment, such as `wipe_desk` where a wiping trajectory is needed to remove the dirt from a desk, and `open_fridge` where a local trajectory needs to adhere to the kinematic constraint when the robot is grasping the door handle. Most tasks in RLBench can be reasonably solved with only macro-action prediction and motion planners. This set of tasks we consider highlights the limitation of these approaches.

Results We train single-task ChainedDiffuser and the baselines. For *Auto-λ*, *HiveFormer* and *InstructRL* we use the numbers reported in the corresponding papers. We show quantitative results in Tables 7.1 and 7.2. ChainedDiffuser consistently achieves better performance than prior methods on all task categories. On the set of challenging tasks for motion planners, ChainedDiffuser gives a significant boost of 60% on average. ChainedDiffuser improves upon open-loop trajectory diffusion model, which demonstrates that delegating global macro-action prediction to a high-level policy to guide local trajectory diffusion helps. *Act3D+ trajectory regression* struggles where multi-modal trajectories are present in demonstrations, e.g. `cup_in_cabinet` where multimodal trajectories exist for grasping the cup and feeding into the cabinet in the training set. This demonstrates that modeling trajectory generation as a multi-step denoising process is advantageous over regression-based model, which aligns with conclusions from previous literature [38].

7.4.2 Real-world Experiments

We conduct experiments with a real-world setup, using a Franka Emika Panda robot with a parallel-jaw gripper. We use a *single* Azure Kinect camera to collect front-view RGB-D image input. See Appendix F.2 for more details on our hardware and data collection setup. We design 7 tasks that involve multi-step actions and continuous interactions with the scene (5 are shown in Figure 7.2), collected $10 - 20$ demos for each tasks, and train a multi-task ChainedDiffuser for real-world deployment. We refer the reader to our supplementary video for qualitative executions of the robot. We evaluate it on 10 episodes for each task, and report success rates in Table 7.3. ChainedDiffuser is able to perform reasonably well on most of the tasks, even for tasks with multiple action modes and skills. The most common failure case is caused by noisy depth image: we leverage point selection for macro-action prediction, which would suffer from incorrect depth estimation in the real world. This could potentially be resolved by more accurate camera calibration with a multi-view camera setup and learning to recover from noisy input, which we leave as our future work.

Task	# Train	Success
`put mask in kit`	20	6/10
`fold and wipe coffee`	20	8/10
`stack cups`	15	7/10
`spread dough`	15	7/10
`fold and wip beans`	20	6/10
`put nails in box`	20	6/10
`press stapler`	10	10/10

Table 7.3: Real-world tasks.

7.4.3 Limitations

Our method currently has the following limitations: **1)** Our trajectory diffuser is conditioned on end-effector poses in SE(3) space. It would be ideal to extend it to full joint configuration space for more flexible trajectory prediction. **2)** Our model performs closed-loop control on the macro-action level, which restricts its flexibility in highly dynamic environments. That said, our framework can be easily extended with closed-loop re-planning at the micro-action level, making the policy more robust to environment dynamics, which we leave as our future work. **3)** Following the standard setting in RLBench, our method assumes access to calibrated cameras. We believe this assumption is valid as mobile robots performing household tasks for humans in the future should have cameras attached to the robots, where these cameras can be calibrated when coming out of the factories.

7.5 Conclusion

We presented ChainedDiffuser, a neural policy architecture for learning 6-DoF robot manipulation from demonstrations. Our model achieves competitive performance on various task settings, in both simulation and the real-world. Our experiments demonstrate that by unifying both transformer-based macro-action detection and diffusion-based trajectory generation, ChainedDiffuser achieves the best of both families and addresses their respective limitations. ChainedDiffuser outperforms both keyframe prediction methods and trajectory diffusion alone, which justifies their unification in our framework. It sets a new state-of-the-art in RLbench, and especially improves performance on contact-rich tasks and tasks that involve articulated objects, where methods that rely on hand designed planners typically struggle.

7.6 Acknowledgements

This work is supported by Sony AI, NSF award No 1849287, DARPA Machine Common Sense, an Amazon faculty award, and an NSF CAREER award.

Chapter 8

3D Diffuser Actor: Policy Diffusion with 3D Scene Representations

ChainedDiffuser unifies non-generative keypose prediction and diffusio-based trajectory planning. A natural question is whether the two objectives can be combined and jointly optimized - keypose prediction is also multimodal and should also benefit from generative modeling. This motivated the development of 3D Diffuser Actor, the first end-to-end 3D diffusion policy for manipulation.

8.1 Introduction

Many robot manipulation tasks are inherently multimodal: at any point during task execution, there may be multiple actions which yield task-optimal behavior. Indeed, human demonstrations often contain diverse ways that a task can be accomplished. A natural choice is then to treat policy learning as a distribution learning problem: instead of representing a policy as a deterministic map $\pi_\theta(x)$, learn the entire distribution of actions conditioned on the current robot state $p(y|x)$ [108, 116, 307, 336].

Recent works use diffusion objectives for learning such state-conditioned action distributions for robot manipulation policies from demonstrations [38, 261, 286]. They outperform deterministic or other alternatives, such as variational autoencoders [228],

This chapter is based on the paper previously published at CoRL 2024 [165]

mixture of Gaussians [37], combination of classification and regression objectives [307], or energy-based objectives [81]. They typically use either low-dimensional (oracle) states [261] or 2D images [38] as their scene representation.

At the same time, 3D robot policies build scene representations by “lifting” features from perspective views to a 3D robot workspace based on sensed depth and camera extrinsics [91, 95, 122, 140, 312, 396]. They have shown to generalize better than 2D robot policies across camera viewpoints and to handle novel camera viewpoints at test time [91, 95]. We conjecture this improved performance comes from the fact that the visual scene tokens and the robot’s actions interact in a common 3D space, that is robust to camera viewpoints, while in 2D policies the neural network need to learn the 2D-to-3D mapping implicitly.

In this work, we marry diffusion for handling multimodality in action prediction with 3D scene representations for effective spatial reasoning. We propose 3D Diffuser Actor, a novel 3D denoising policy transformer that takes as input a tokenized 3D scene representation, a language instruction and a noised end-effector’s future translation and rotation trajectory, and predicts the error in translations and rotations for the robot’s end-effector. The model represents both the scene tokens and the end-effector locations in the same 3D space and fuses them with relative-position 3D attentions [309, 325], which achieves translation equivariance and helps generalization.

We test 3D Diffuser Actor in learning robot manipulation policies from demonstrations on the simulation benchmarks of RLBench [139] and CALVIN [236], as well as in the real world. 3D Diffuser Actor sets a new state-of-the-art on RLBench with a 18.1% absolute gain on multi-view setups and 13.1% on single-view setups, outperforming existing 3D policies and 2D diffusion policies. On CALVIN, it outperforms the current SOTA in the setting of zero-shot unseen scene generalization by a 9% relative gain. We further show 3D Diffuser Actor can learn multi-task manipulation in the real world across 12 tasks from a handful of real-world demonstrations. We empirically show that 3D Diffuser Actor outperforms all existing policy formulations, which either do not use 3D scene representations, or do not use action diffusion. We further compare against ablative versions of our model and show the importance of the 3D relative attentions.

Our contributions: The main contribution of this work is to combine 3D scene representations and diffusion objectives for learning robot policies from demonstrations. 3D robot policies have not yet been combined with diffusion objectives. An exception is ChainedDiffuser [364], that uses diffusion models as a drop-in replacement for motion planners, rather than manipulation policies, since it relies on other learning-based policies (Act3D [91]) to supply the target 3D keypose to reach. We compare against ChainedDiffuser in our experiments and show we greatly outperform it.

Concurrent work: Concurrent to our effort, 3D diffusion policy [393] shares a similar goal of combining 3D representations with diffusion objectives for learning manipulation from demonstrations. Though the two works share the same goal, they have very different architectures. Unlike 3D Diffuser Actor, the model of [393] does not condition on a tokenized 3D scene representation but rather on a holistic 1D embedding pooled from the 3D scene point cloud. We compare 3D Diffuser Actor against [393] in our experiments and show it greatly outperforms it. We believe this is because tokenized scene representations, used in 3D Diffuser Actor, are robust to scene changes: if a part of the scene changes, only the corresponding subset of 3D scene tokens is affected. In contrast, holistic scene embeddings pooled across the scene are always affected by any scene change. Thanks to this spatial disentanglement of the 3D scene tokenization, 3D Diffuser Actor generalizes better.

Our models, code and videos of our manipulation results are available at `http://3d-diffuser-actor.github.io/`.

8.2 Related Work

Learning manipulation policies from demonstrations Earlier works on learning from demonstrations train deterministic policies with behavior cloning [16, 266]. To better handle action multimodality, approaches discretize action dimensions and use cross entropy losses [225, 312, 396]. Generative adversarial networks [61, 116, 336], variational autoencoders [228], combined Categorical and Gaussian distributions [46, 103, 307] and Energy-Based Models (EBMs) [81, 93, 330] have been used to learn from multimodal demonstrations. Diffusion models [117, 320] are a powerful class of generative models related to EBMs in that they model the score of the distribution, else, the gradient of the energy, as opposed to the energy itself [299, 317]. The key idea behind diffusion models is to iteratively transform a simple prior distribution into a target distribution by applying a sequential denoising process. They have been used for modeling state-conditioned action distributions in imitation learning [241, 261, 286, 296, 340, 355] from low-dimensional input, as well as from visual sensory input, and show both better mode coverage and higher fidelity in action prediction than alternatives.

Diffusion models in robotics Beyond policy representations in imitation learning, diffusion models have been used to model cross-object and object-part arrangements [76, 93, 209, 241, 316], visual image subgoals [5, 14, 50, 160], and offline reinforcement learning [26, 106, 376]. ChainedDiffuser [364] proposes to replace motion planners, commonly used for keypose to keypose linking, with a trajectory diffusion model that conditions on the 3D scene feature cloud and the predicted target 3D keypose to denoise a trajectory from the current to the target keypose. It uses a diffusion model that takes as input 3D end-effector keyposes predicted by Act3D [91] and a 3D representation of the scene to infer robot end-effector trajectories that link the current end-effector pose to the predicted one. 3D Diffuser Actor instead predicts the next 3D keypose for the robot's end-effector alongside the linking trajectory, which is a much harder task than linking two given keyposes. 3D Diffusion Policy [393] also combines 3D scene representations with diffusion objectives but uses 1D point cloud embeddings. We compare against both ChainedDiffuser and 3D Diffusion Policy in our experimental section and show we greatly outperform them.

2D and 3D scene representations for robot manipulation End-to-end image-to-action policy models, such as RT-1 [19], RT-2 [20], GATO [284], BC-Z [141], RT-X [255], Octo [253] and InstructRL [199] leverage transformer architectures for the direct prediction of 6-DoF end-effector poses from 2D video input. However, this approach comes at the cost of requiring thousands of demonstrations to implicitly model 3D geometry and adapt to variations in the training domains. 3D scene-to-action policies, exemplified by C2F-ARM [140] and PerAct [312], involve voxelizing the robot's workspace and learning to identify the 3D voxel containing the next end-effector keypose. However, this becomes computationally expensive as resolution requirements increase. Consequently, related approaches resort to either coarse-to-fine voxelization, equivariant networks [122] or efficient attention operations [133] to mitigate computational costs. Act3D [91] foregoes 3D scene voxelization altogether; it instead computes a 3D action map of variable spatial resolution by sampling 3D points in the empty workspace and featurizing them using cross-attentions to the 3D physical scene points. Robotic View Transformer (RVT) [95] re-projects the input RGB-D image to alternative image views, featurizes those and lifts the predictions to 3D to infer 3D locations for the robot's end-effector.

3D Diffuser Actor builds upon works of Act3D [91] from 3D policies and upon the works of [38, 340] from diffusion policies. It uses a tokenized 3D scene representation, similar to [91], but it is a probabilistic model instead of a deterministic one. It does not sample 3D points and does not infer 3D action maps. It uses diffusion objectives instead of classification or regression objectives used in [91]. In contrast to [316, 340], it uses 3D scene representations instead of 2D images or low-dimensional states. We compare against both 2D diffusion policies and 3D policies in our experiments and show 3D Diffuser Actor greatly outperforms them. We highlight the differences between our model and related models in Figure G.2 in the Appendix, and we refer to Figures G.3 and G.4 for more architectural details of 3D Diffuser Actor and Act3D.

8.3 Method

3D Diffuser Actor is trained to imitate demonstration trajectories of the form of $\{(\mathbf{o}_1, \mathbf{a}_1), (\mathbf{o}_2, \mathbf{a}_2), ...\}$, accompanied with a task language instruction l, similar to previous works [91, 138, 312, 359], where $\mathbf{o}_t$ stands for the visual observation and $\mathbf{a}_t$ stands for robot action at timestep t. Each observation $\mathbf{o}_t$ is one or more posed RGB-D images. Each action $\mathbf{a}_t$ is an end-effector pose and is decomposed into 3D location, rotation and binary (open/close) state: $\mathbf{a}_t = \{\mathbf{a}_t^{\text{loc}} \in \mathbb{R}^3, \mathbf{a}_t^{\text{rot}} \in \mathbb{R}^6, \mathbf{a}_t^{\text{open}} \in \{0, 1\}\}$. We represent rotations using the 6D rotation representation of [411] for all environments in all our experiments, to avoid the discontinuities of the quaternion representation. We will use the notation $\boldsymbol{\tau}_t = (\mathbf{a}_{t:t+T}^{\text{loc}}, \mathbf{a}_{t:t+T}^{\text{rot}})$ to denote the trajectory of 3D locations and rotations at timestep t, of temporal horizon T. Our model, at each timestep t predicts a trajectory $\boldsymbol{\tau}_t$ and binary states $\mathbf{a}_{t:t+T}^{\text{open}}$.

The architecture of 3D Diffuser Actor is shown in Figure 8.1. It is a conditional diffusion probabilistic model [118, 320] of trajectories given the visual scene and a language instruction; it predicts a whole trajectory $\boldsymbol{\tau}$ at once, non autoregressively, through iterative denoising, by inverting a process that gradually adds noise to a sample $\boldsymbol{\tau}^0$. The diffusion process is associated with a variance schedule $\{\beta^i \in (0, 1)\}_{i=1}^N$,

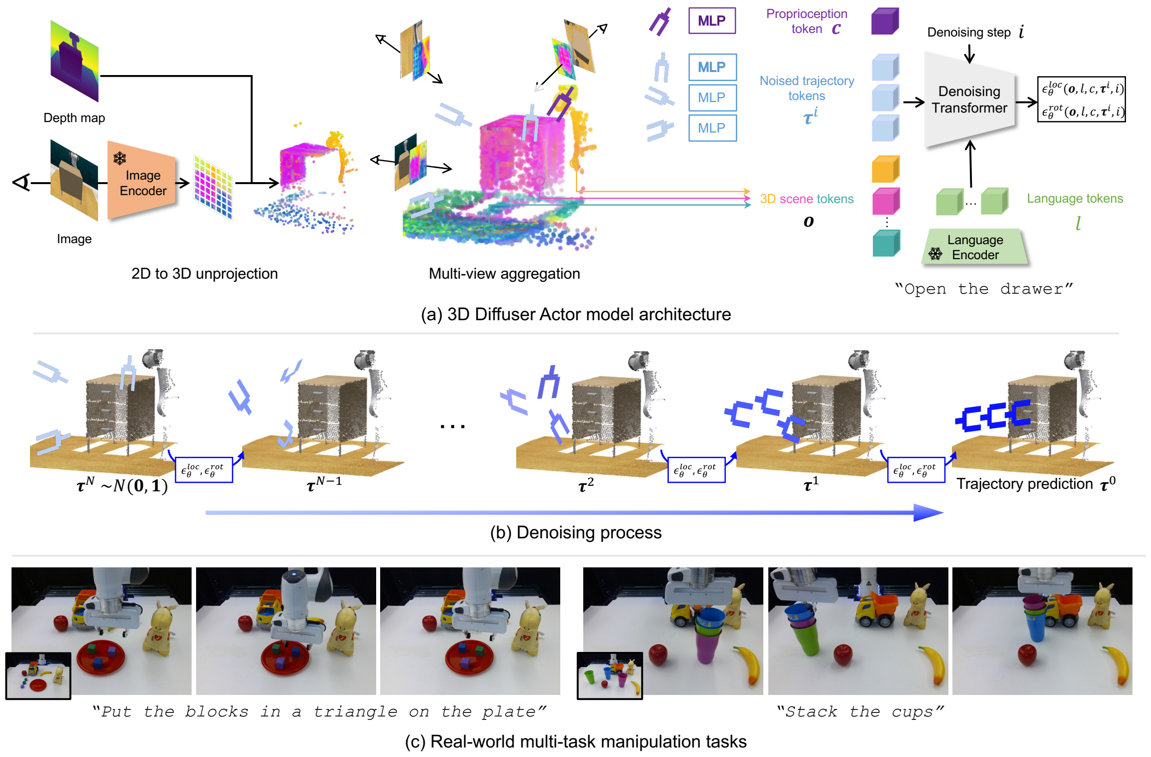

Depth map
Image
Image Encoder
2D to 3D unprojection
Multi-view aggregation
MLP
MLP
MLP
MLP
Proprioception token c
Noised trajectory tokens τ^i
3D scene tokens o
Denoising step i
Denoising Transformer
$\epsilon_\theta^{loc}(o, l, c, \tau^i, i)$
$\epsilon_\theta^{rot}(o, l, c, \tau^i, i)$
Language tokens l
Language Encoder
"Open the drawer"
(a) 3D Diffuser Actor model architecture
$\tau^N \sim N(\mathbf{0}, \mathbf{1})$
$\epsilon_\theta^{loc}, \epsilon_\theta^{rot}$
τ^{N-1}
τ^2
τ^1
Trajectory prediction τ^0
(b) Denoising process
"Put the blocks in a triangle on the plate"
"Stack the cups"
(c) Real-world multi-task manipulation tasks

Figure 8.1: **Overview of 3D Diffuser Actor.** **(a)** 3D Diffuser Actor is a conditional diffusion probabilistic model of robot 3D trajectories. At diffusion step i, it converts the current noised estimate of the robot's future action trajectory $\boldsymbol{\tau}^i$, the posed RGB-D views $\mathbf{o}$, and proprioceptive information c to a set of 3D tokens. It fuses information among these 3D tokens and language instruction tokens l using 3D relative denoising transformers to predict the noise of 3D robot locations $\epsilon_\theta^{loc}(\mathbf{o}, l, c, \boldsymbol{\tau}^i, i)$ and the noise of 3D robot rotations $\epsilon_\theta^{rot}(\mathbf{o}, l, c, \boldsymbol{\tau}^i, i)$. **(b)** During inference, 3D Diffuser Actor iteratively denoises the noised estimate of the robot's future trajectory. **(c)** 3D Diffuser Actor works in the real world and captures the multiple behavioural modes in the training demonstrations.

which defines how much noise is added at each diffusion step. The noisy version of sample $\boldsymbol{\tau}^0$ at diffusion step i can then be written as $\boldsymbol{\tau}^i = \sqrt{\bar{\alpha}^i}\boldsymbol{\tau}^0 + \sqrt{1-\bar{\alpha}^i}\epsilon$, where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ is a sample from a Gaussian distribution (with the same dimensionality as $\boldsymbol{\tau}^0$), $\alpha^i = 1 - \beta^i$, and $\bar{\alpha}^i = \prod_{j=1}^{i} \alpha^j$.

3D Relative Denoising Transformer 3D Diffuser Actor models a learned gradient of the denoising process with a 3D relative transformer $\hat{\epsilon} = \epsilon_\theta(\boldsymbol{\tau}_t^i; i, \mathbf{o}_t, l, c_t)$ that takes as input the noisy trajectory $\boldsymbol{\tau}_t^i$ at timestep t, diffusion step i, and conditioning information from the language instruction l, the visual observation $\mathbf{o}_t$ and proprioception c_t of timestep t, to predict the noise component $\hat{\epsilon}$. At each timestep t and diffusion step i, we convert visual observations $\mathbf{o}_t$, proprioception c_t and noised trajectory estimate $\boldsymbol{\tau}_t^i$ to a set of 3D tokens. Each 3D token is represented by a latent embedding and a 3D position. Our model fuses all 3D tokens using relative 3D attentions, and additionally fuses information from the language instruction using normal attentions, since it is meaningless to define 3D coordinates for language tokens. Next, we describe how we featurize each piece of input (when not ambiguous, we omit the subscript t for clarity).

3D tokenization At each diffusion step i, we represent the noisy estimate $\boldsymbol{\tau}^i$ of the clean trajectory $\boldsymbol{\tau}^0$ as a sequence of 3D trajectory tokens, by mapping each noisy pose $\mathbf{a}^i$ of $\boldsymbol{\tau}^i$, to a latent embedding vector with a MLP, and a 3D position which comes from the noised 3D translation component $\mathbf{a}^{loc,i}$. We featurize each image view using a 2D feature extractor and obtain a corresponding 2D feature map $F \in \mathbb{R}^{H \times W \times c}$, where c is the number of feature channels and H, W the spatial dimensions, using a pre-trained CLIP ResNet50 2D image encoder [276]. Given the corresponding depth map from that view, we compute the 3D location (X, Y, Z) for each of the $H \times W$ feature patches by averaging the depth value inside the patch extent. We map the pixel coordinate and depth value of a patch to a corresponding 3D coordinate using camera intrinsics and extrinsics and the pinhole camera model. This results in a 3D scene token set of cardinality $H \times W$. Each scene token is represented by the corresponding patch feature vector $F_{x,y}$, the feature vector at the corresponding patch coordinate (x, y), and a 3D position. If more than one views are available, we aggregate 3D scene tokens from each, to obtain the final 3D scene token set. The proprioception c is also a 3D scene token, with a learnable latent representation and the 3D positional embedding corresponding to the end-effector's current 3D location. Lastly, we map the language task instruction to language tokens using a pre-trained CLIP language encoder, following previous works [91].

Our 3D Relative Denoising Transformer applies relative self-attentions among all 3D tokens, and cross-attentions to the language tokens. For the 3D self-attentions, we use the rotary positional embeddings [325] to encode relative positional information in attention layers. The attention weight between query q and key k is written as: $e_{q,k} \propto \mathbf{x}_q^T \mathbf{M}(\mathbf{p}_q - \mathbf{p}_k)\mathbf{x}_k$, where $\mathbf{x}_q$ / $\mathbf{x}_k$ and $\mathbf{p}_q$ / $\mathbf{p}_k$ denote the features and

3D positions of the query / key, and $\mathbf{M}$ is a matrix function which depends only on the relative positions of the query and key, inspired by recent work in visual correspondence [94, 186] and 3D manipulation [91, 364]. We feed the final trajectory tokens to MLPs to predict: (1) the noise $\epsilon_\theta^{loc}(\mathbf{o}, l, c, \boldsymbol{\tau}^i, i)$ and $\epsilon_\theta^{rot}(\mathbf{o}, l, c, \boldsymbol{\tau}^i, i)$ added to $\boldsymbol{\tau}^0$'s sequence of 3D translations and 3D rotations, respectively, and (2) the end-effector opening $f_\theta^{\text{open}}(\mathbf{o}, l, c, \boldsymbol{\tau}^i, i) \in [0,1]^T$.

Training and inference During training, we randomly sample a time step t and a diffusion step i and add noise $\epsilon = (\epsilon^{\text{loc}}, \epsilon^{\text{rot}})$ to a ground-truth trajectory $\boldsymbol{\tau}_t^0$. We use the $L1$ loss for reconstructing the sequence of 3D locations and 3D rotations. We use binary cross-entropy (BCE) loss to supervise the end-effector opening f_θ^{open}, we use the prediction from i=1 at inference time. Our objective reads:

$$\mathcal{L}_\theta = w_1 \|(\epsilon_\theta^{\text{loc}}(\mathbf{o}, l, c, \boldsymbol{\tau}^i, i) - \epsilon^{\text{loc}}\| + w_2 \|(\epsilon_\theta^{\text{rot}}(\mathbf{o}, l, c, \boldsymbol{\tau}^i, i) - \epsilon^{\text{rot}}\| + \text{BCE}(f_\theta^{\text{open}}(\mathbf{o}, l, c, \boldsymbol{\tau}^i, i), \mathbf{a}_{1:T}^{\text{open}}),$$

where w_1, w_2 are hyperparameters estimated using cross-validation. To draw a sample from the learned distribution $p_\theta(\boldsymbol{\tau}|\mathbf{o}, l, c)$, we start by drawing a sample $\boldsymbol{\tau}_N \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$. Then, we progressively denoise the sample by iterated application of ϵ_θ N times according to a specified sampling schedule [117, 321], which terminates with $\boldsymbol{\tau}_0$ sampled from $p_\theta(\boldsymbol{\tau})$: $\boldsymbol{\tau}^{i-1} = \frac{1}{\sqrt{\alpha^i}}\left(\boldsymbol{\tau}^i - \frac{\beta^i}{\sqrt{1-\bar{\alpha}_i}}\epsilon_\theta(\mathbf{o}, l, c, \boldsymbol{\tau}^i, i)\right) + \frac{1-\bar{\alpha}^{i+1}}{1-\bar{\alpha}^i}\beta^i\mathbf{z}$, where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ is a random variable of appropriate dimension. Empirically, we found that using separate noise schedulers for $\mathbf{a}^{\text{loc}}$ and $\mathbf{a}^{\text{rot}}$, specifically, scaled-linear and square cosine scheduler, respectively, achieves better performance.

Implementation details At training, we segment demonstration trajectories at detected end-effector *keyposes*, such as a change in the open/close end-effector state or local extrema of velocity/acceleration, following previous works [91, 138, 199, 312]. We then resample each trajectory segment to have the same length T. During inference, 3D Diffuser Actor can either predict and execute the full trajectory of actions up to the next keypose (including the keypose), or just predict the next keypose and use a sampling-based motion planner to reach it, similar to previous works [91, 103, 312].

Due to space limits, please check Section G.2.3 for detailed model diagram, Section G.2.5 for our choice of hyper-parameters, Section G.2.6 for detailed formulation of denoising diffusion probabilistic models and Section G.2.7 for discussion of noise schedulers in the Appendix.

8.4 Experiments

We test 3D Diffuser Actor in multi-task manipulation on RLBench [139] and CALVIN [236], two established learning from demonstrations benchmarks, and in the real world.

8.4.1 Evaluation on RLBench

RLBench is built atop the CoppelaSim [291] simulator, where a Franka Panda Robot is used to manipulate the scene. On RLBench, our model and all baselines are trained to predict the next end-effector keypose as opposed to keypose trajectory; all methods employ a low-level motion planner BiRRT [176], native to RLBench, to reach the predicted robot keypose. We train and evaluate 3D Diffuser Actor on two setups based on the number of available cameras: **1.** *Multi-view setup*, introduced in [312], that uses a suite of 18 manipulation tasks, each with 2-60 variations, which concern scene variability across object poses, appearance and semantics. There are four RGB-D cameras available, front, wrist, left shoulder and right shoulder. The wrist camera moves during manipulation. **2.** *Single-view setup*, introduced in [391], that uses a suite of 10 manipulation tasks. Only the front RGB-D camera view is available. We evaluate policies by task completion success rate, which is the proportion of execution trajectories that achieve the goal conditions specified in the language instructions [91, 312].

Baselines All compared methods on RLBench are 3D policies that use depth and camera parameters during featurization of the RGB-D input. We compare against the following: C2F-ARM-BC [140] and PerAct [312] that voxelize the 3D workspace, Hiveformer [103] that featurizes XYZ coordinates aligned with the 2D RGB frames, PolarNet [29] that featurizes a scene 3D point cloud, GNFactor [391] that uses a single RGB-D view and is trained to complete the 3D feature volume, RVT [95] and Act3D [91], that are the previous SOTA methods on RLBench. We report results for RVT, PolarNet and GNFactor based on their respective papers. Results for CF2-ARM-BC and PerAct are presented as reported in [95]. Results for Hiveformer are copied from [29]. We retrained Act3D on the multi-view setup using the publicly available code,

	Avg. Success ↑	open drawer	slide block	sweep to dustpan	meat off grill	turn tap	put in drawer	close jar	drag stick	stack blocks	screw bulb	put in safe	place wine	put in cupboard	sort shape	push buttons	insert peg	stack cups	place cups
C2F-ARM-BC [140]	20.1	20	16	0	20	68	4	24	24	0	8	12	8	0	8	72	4	0	0
PerAct [312]	49.4	88.0$_{\pm 5.7}$	74.0$_{\pm 13.0}$	52.0$_{\pm 0.0}$	70.4$_{\pm 2.0}$	88.0$_{\pm 4.4}$	51.2$_{\pm 4.7}$	55.2$_{\pm 4.7}$	89.6$_{\pm 4.1}$	26.4$_{\pm 3.2}$	17.6$_{\pm 2.0}$	86.0$_{\pm 3.6}$	44.8$_{\pm 7.8}$	28.0$_{\pm 4.4}$	16.8$_{\pm 4.7}$	92.8$_{\pm 3.0}$	5.6$_{\pm 4.1}$	2.4$_{\pm 2.2}$	2.4$_{\pm 3.2}$
HiveFormer [103]	45	52	64	28	**100**	80	68	52	76	8	8	76	80	32	8	84	0	0	0
PolarNet [29]	46	84	56	52	**100**	80	32	36	92	4	44	84	40	12	12	96	4	8	0
RVT [95]	62.9	71.2$_{\pm 6.9}$	81.6$_{\pm 5.4}$	72.0$_{\pm 0.0}$	88.0$_{\pm 2.5}$	93.6$_{\pm 4.1}$	88.0$_{\pm 5.7}$	52.0$_{\pm 2.5}$	99.2$_{\pm 1.6}$	28.8$_{\pm 3.9}$	48.0$_{\pm 5.7}$	91.2$_{\pm 3.0}$	91.0$_{\pm 5.2}$	49.6$_{\pm 3.2}$	36.0$_{\pm 2.5}$	**100.0**$_{\pm 0.0}$	11.2$_{\pm 3.0}$	26.4$_{\pm 8.2}$	4.0$_{\pm 2.5}$
Act3D [91]	63.2	78.4$_{\pm 11.2}$	96.0$_{\pm 2.5}$	**86.4**$_{\pm 6.5}$	95.2$_{\pm 1.6}$	94.4$_{\pm 2.0}$	91.2$_{\pm 6.9}$	**96.8**$_{\pm 3.0}$	80.8$_{\pm 6.4}$	4.0$_{\pm 3.6}$	32.8$_{\pm 6.9}$	95.2$_{\pm 4.0}$	59.2$_{\pm 9.3}$	67.2$_{\pm 3.0}$	29.6$_{\pm 3.2}$	93.6$_{\pm 2.0}$	24.0$_{\pm 8.4}$	9.6$_{\pm 6.0}$	3.2$_{\pm 3.0}$
3D Diffuser Actor (ours)	**81.3**	**89.6**$_{\pm 4.1}$	**97.6**$_{\pm 3.2}$	84.0$_{\pm 4.4}$	96.8$_{\pm 1.6}$	**99.2**$_{\pm 1.6}$	**96.0**$_{\pm 3.6}$	**96.0**$_{\pm 2.5}$	**100.0**$_{\pm 0.0}$	**68.3**$_{\pm 3.3}$	**82.4**$_{\pm 2.0}$	**97.6**$_{\pm 2.0}$	**93.6**$_{\pm 4.8}$	**85.6**$_{\pm 4.1}$	**44.0**$_{\pm 4.4}$	98.4$_{\pm 2.0}$	**65.6**$_{\pm 4.1}$	**47.2**$_{\pm 8.5}$	**24.0**$_{\pm 7.6}$

Table 8.1: **Evaluation on RLBench on the multi-view setup.** We show the mean and standard deviation of success rates average across all random seeds. 3D Diffuser Actor outperforms all prior arts on most tasks by a large margin. Variances are included when available.

	Avg. Success ↑	close jar	open drawer	sweep to dustpan	meat off grill	turn tap	slide block	put in drawer	drag stick	push buttons	stack blocks
GNFactor [391]	31.7	25.3	76.0	28.0	57.3	50.7	20.0	0.0	37.3	18.7	4.0
Act3D [91]	65.3	52.0$_{\pm 5.7}$	84.0$_{\pm 8.6}$	80.0$_{\pm 9.8}$	66.7$_{\pm 1.9}$	64.0$_{\pm 5.7}$	**100.0**$_{\pm 0.0}$	54.7$_{\pm 3.8}$	86.7$_{\pm 1.9}$	64.0$_{\pm 1.9}$	0.0$_{\pm 0.0}$
3D Diffuser Actor	**78.4**	**82.7**$_{\pm 1.9}$	**89.3**$_{\pm 7.5}$	**94.7**$_{\pm 1.9}$	**88.0**$_{\pm 5.7}$	**80.0**$_{\pm 8.6}$	92.0$_{\pm 0.0}$	**77.3**$_{\pm 3.8}$	**98.7**$_{\pm 1.9}$	**69.3**$_{\pm 5.0}$	**12.0**$_{\pm 3.7}$

Table 8.2: **Evaluation on RLBench on the single-view setup.** 3D Diffuser Actor outperforms prior state-of-the-art baselines, GNFactor and Act3D, on most tasks by a large margin.

because we found some differences on the train and test splits used in the original paper. We also trained Act3D on the single-view setup to use as additional baseline for the single-view setup, alongside GNFactor. 3D Diffuser Actor and all baselines are trained on the same set of keyposes extracted from expert demonstrations [138].

We show quantitative results for the multi-view setup in Table 8.1 and for single-view setup in Table 8.2. On multi-view, 3D Diffuser Actor achieves an average 81.3% success rate among all 18 tasks, an absolute improvement of +18.1% over Act3D, the previous state-of-the-art. In particular, 3D Diffuser Actor achieves big leaps on long-horizon high-precision tasks with multiple modes, such as *stack blocks*, *stack cups* and *place cups*, which most baselines fail to complete. All baselines use classification or regression losses, our model is the only one to use diffusion for action prediction.

On single-view, 3D Diffuser Actor outperforms GNFactor by +46.7% and Act3D

	Avg. Success.
2D Diffuser Actor	47.0
3D Diffuser Actor w/o RelAttn.	71.3
3D Diffuser Actor (ours)	**81.3**

Table 8.3: **Ablation study.** Our model significantly outperforms its counterparts that do not use 3D scene representations or translation-equivariant 3D relative attention.

by +13.1%. Surprisingly, Act3D also outperforms GNFactor by a big margin. This suggests that the choice of 3D scene representation is more crucial than 3D feature completion. Since Act3D has a similar 3D scene tokenization as 3D Diffuser Actor, this shows the importance of diffusion over alternatives to handle multimodality in prediction, specifically over sampling based 3D action maps and rotation regression.
Ablations We consider the following ablative versions of our model: **1.** 2D Diffuser Actor, our implementation of 2D Diffusion Policy of [38]. We remove the 3D scene encoding from 3D Diffuser Actor and instead use per-image 2D representations by average-pooling features within each view. We add learnable embeddings to distinguish between different views and fuse them with action estimates, as done in [38]. **2.** 3D Diffuser Actor w/o RelAttn., an ablative version that uses absolute (non-relative) attentions.

We show ablative results in Table 8.3. 3D Diffuser Actor largely outperforms its 2D counterpart, 2D Diffuser Actor. This shows the importance of 3D scene representations in performance. 3D Diffuser Actor with absolute 3D attentions (3D Diffuser Actor w/o RelAttn.) is worse than 3D Diffuser Actor with relative 3D attentions. This shows that translation equivariance through relative attentions is important for generalization. Despite that, this ablative version already outperforms all prior arts in Table 8.1, proving the effectiveness of marrying 3D representations and diffusion policies.

8.4.2 Evaluation on CALVIN

The CALVIN benchmark is built on top of the PyBullet [45] simulator and involves a Franka Panda Robot arm that manipulates the scene. CALVIN consists of 34 tasks and 4 different environments (A, B, C and D). All environments are equipped with a desk, a sliding door, a drawer, a button that turns on/off an LED, a switch that controls a lightbulb and three different colored blocks (red, blue and pink). These environments differ from each other in the texture of the desk and positions of the objects. CALVIN provides 24 hours of tele-operated unstructured play data, 35% of which are annotated with language descriptions. Each instruction chain includes five language instructions that need to be executed sequentially. We evaluate on the so called *zero-shot generalization setup*, where models are trained in environments A, B and C and tested in D. We report the success rate and the average number of completed sequential tasks, following previous works [235, 359]. No motion planner is available in CALVIN so all models need to predict robot pose trajectories.

Baselines All methods tested so far in CALVIN are 2D policies, that do not use depth or camera extrinsics. We compare against the hierarchical 2D policies of MCIL [226], HULC [235] and SuSIE [14] which predict latent features or images of subgoals given a language instruction, which they feed into lower-level subgoal-conditioned policies. They can train the low-level policy on all data available in CALVIN as opposed to the language annotated subset only. We compare against large scale 2D transformer policies of RT-1 [19], RoboFlamingo [184] and GR-1 [359] which pretrain on large amounts of interaction or observational (video alone) data. We report results for HULC, RoboFlamingo, SuSIE and GR-1 from the respective papers. Results from MCIL are borrowed from [235]. Results from RT-1 are copied from [359].

We also compare against 3D Diffusion Policy [393] and ChainedDiffuser [364], which we train both on the language annotated training set. We evaluate 3D Diffusion Policy, ChainedDiffuser and 3D Diffuser Actor with the final checkpoints across 3 seeds. We report both the mean and standard deviation of evaluation results. We devise an algorithm to extract keyposes on CALVIN, since prior works do not use keyposes. We define keyposes as those at frames with significant motion change. Both ChainedDiffuser and 3D Diffuser Actor segment the demonstations based on the keyposes. Notably, though keyposes extracted in RLBench have clear structure since they correspond to programmatically labeled 3D waypoints [138, 139], keyposes extracted in CALVIN are instead noisy and random, as the benchmark consists of human play trajectories.

For evaluation, prior works [14, 184, 235] predict a maximum temporal horizon of 360 actions to complete each task, while, on average, ground-truth trajectories are only 60 actions long. Our model predicts trajectories and replans after each trajectory

	Train episodes	Task completed in a row					
		1	2	3	4	5	Avg. Len
3D Diffusion Policy [393]	Lang	$28.7_{\pm 0.4}$	$2.7_{\pm 0.4}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.31_{\pm 0.04}$
MCIL [226]	All	30.4	1.3	0.2	0.0	0.0	0.31
HULC [235]	All	41.8	16.5	5.7	1.9	1.1	0.67
RT-1 [19]	Lang	53.3	22.2	9.4	3.8	1.3	0.90
ChainedDiffuser [364] (60 keyposes)	Lang	$49.9_{\pm 0.01}$	$21.1_{\pm 0.01}$	$8.0_{\pm 0.01}$	$3.5_{\pm 0.0}$	$1.5_{\pm 0.0}$	$0.84_{\pm 0.02}$
RoboFlamingo [184]	Lang	82.4	61.9	46.6	33.1	23.5	2.48
SuSIE [14]	All	87.0	69.0	49.0	38.0	26.0	2.69
GR-1 [359]	Lang	85.4	71.2	59.6	49.7	40.1	3.06
3D Diffuser Actor (ours)	Lang	**93.8**$_{\pm 0.01}$	**80.3**$_{\pm 0.0}$	**66.2**$_{\pm 0.01}$	**53.3**$_{\pm 0.02}$	**41.2**$_{\pm 0.01}$	**3.35**$_{\pm 0.04}$

Table 8.4: **Zero-shot long-horizon evaluation on CALVIN** on 3 random seeds.

close box	put duck	insert peg into hole	insert peg into torus	put mouse	open pen
100	100	50	30	80	100
press stapler	put grapes	sort rectangle	stack blocks	stack cups	put block in triangle
90	90	50	20	40	90

Table 8.5: **Multi-Task performance on real-world tasks.**

has been executed, instead of after each individual actions. On average, it takes 10 trajectory inference steps to complete a task. We thus allow our model to predict a maximum temporal horizon of 60 trajectories for each task.

We show quantitative results in Table 8.4. 3D Diffuser Actor outperforms the state-of-the-art. ChainedDiffuser does not work well on this benchmark. The reason is that its deterministic keypose prediction module fails to predict end-effector keyposes accurately due to multimodality present in human demonstrations of CALVIN in comparison to programmatically collected demonstrations of RLBench. Allowing 3D Diffuser Actor to have a longer horizon before terminating increases the performance significantly, which suggests that the model learns to retry under failure.

8.4.3 Evaluation in the real world

We validate 3D Diffuser Actor in learning manipulation tasks from real-world demonstrations across 12 tasks. We use a Franka Emika robot equipped with a Azure

Kinect RGB-D sensor at a front view. Images are originally captured at 1280×720 resolution and downsampled to a resolution of 256×256. During inference, we utilize the BiRRT [176] planner provided by the MoveIt! ROS package [44] to reach the predicted poses.

We collect 15 demonstrations per task, most of which naturally contain noise and multiple modes of human behavior. For example, we pick one of two ducks to put in the bowl, we insert the peg into one of two holes and we put one of three grapes in the bowl. We evaluate 10 episodes for each task and report the success rate. We show quantitative results in Table 8.5 and video results on our project webpage. 3D Diffuser Actor effectively learns real-world manipulation from a handful of demonstrations.

Runtime We compare the control frequency of our model against ChainedDiffuser [364] and 3D Diffusion Policy [393] on CALVIN, using an NVIDIA 2080 Ti graphic card. The inference speed of 3D Diffuser Actor, ChainedDiffuser and 3D Diffusion Policy is 600ms, 1170ms (50 for keypose detection and 1120 for trajectory optimization) and 581ms for executing 6, 6 and 4 end-effector poses. The respective control frequency is 10 Hz, 5.1 Hz and 5.2 Hz.

Limitations Despite its SOTA performance with large margins over existing methods, our framework currently has the following limitations: **1.** It requires camera calibration and depth information, same as all 3D policies. **2.** All tasks in RLBench and CALVIN are quasi-static. Extending our method to dynamic tasks and velocity control is a direct avenue of future work. **3.** It is on average slower than non-diffusion policies. This can be improved by recent techniques on reducing the inference steps of diffusion models.

Please, refer to our Appendix for more experiments and details: Section G.1.1 for robustness of 3D Diffuser Actor to depth noise, Section G.1.2 for discussion of failure cases, Section G.1.3 and G.1.4 for descriptions of tasks in RLBench and the real world, Section G.1.5 for descriptions of baselines, Section G.1.5 for implementation details of re-training 3D Diffusion Policy on CALVIN, Section G.1.6 for keypose discovery on CALVIN. Video results can be found in our supplementary file.

8.5 Conclusion

We present 3D Diffuser Actor, a manipulation policy that combines 3D scene representations and action diffusion. Our method sets a new state-of-the-art on RLBench and CALVIN by a large margin over existing 2D and 3D policies, and learns robot control in the real world from a handful of demonstrations. 3D Diffuser Actor builds upon recent progress on 3D tokenized scene representations for robotics and on generative models and shows how their combination is a powerful learning from demonstration method. Our future work includes learning 3D Diffuser Actor policies from suboptimal demonstrations and scaling up training data in simulation and in the real world.

Chapter 9

3D Flow Actor: An Efficient Flow Matching 3D Policy for Bimanual and Unimanual Control

While a significant leap for manipulation, 3D Diffuser Actor suffers from long training times and slow inference speed, limiting its widespread adoption as a policy formulation. 3D Flow Actor is a direct drop-in replacement: it drastically increases efficiency without sacrificing performance, it shows wide applicability to bimanual and unimanual control, and it has good scaling properties, which will be explored in our last chapter.

9.1 Introduction

Single-arm manipulation has achieved great success in handling long-horizon and high-precision tasks [139, 185, 198], even in highly cluttered environments [75, 169]. However, the lack of coordination between multiple end effectors largely constrains single-arm systems to simple pick-and-place tasks, making them inadequate for addressing the more complex and diverse manipulation challenges encountered in real-world daily tasks. To overcome these challenges, bimanual systems offer a promising alternative by enabling more dexterous and coordinated interactions with the environment [318].

Although bimanual setups improve the ability of a robot to perform more intricate and dexterous tasks, they also impose stricter demands on spatio-temporal precision. Both arms must operate in a tightly coordinated manner, executing actions in the correct temporal sequence and at precisely aligned spatial locations. This added complexity makes learning effective manipulation policies more difficult than in the single-arm case. Despite growing interest, existing approaches [97, 98, 100, 200, 405] still fall short of achieving robust generalization across a wide range of tasks.

In parallel, recent advances in single-arm manipulation have demonstrated the power of diffusion models in capturing multimodal behaviors [38, 164, 286, 393], achieving high-precision action prediction through 3D scene understanding [91, 95, 312, 364] and impressive generalization to various tasks and language instructions [15, 183, 275]. A natural next step toward robust bimanual manipulation is to integrate these advances. In fact, we show that an adaptation of 3D Diffuser Actor (3DDA) [164] - a model that combines diffusion-based action generation with 3D scene representations - already establishes a new state of the art on the bimanual manipulation benchmark PerAct2 [100]. Next, we ask the question: what prevents 3DDA from being deployed in real-world bimanual manipulation scenarios? Our experimentation suggests two key bottlenecks: slow inference speed and long training time. For instance, on PerAct2, 3DDA requires approximately 21 days to train and operates at 0.5Hz during inference. The prolonged training time significantly limits the model's ability to adapt to new tasks, while the slow inference speed makes it unsuitable for real-time or dynamic task execution.

In light of these observations, we introduce 3D Flow Actor (3DFA), a significantly more efficient extension of 3DDA that improves both training time and inference speed by an order of magnitude. On the PerAct2 bimanual manipulation benchmark [100], 3DFA reduces the training time from 21 days to 16 hours and increases the inference speed from 0.5Hz to 13.3Hz, without sacrificing performance.

To enable faster inference, we replace the DDPM-based [117] formulation used in 3DDA with a Flow Matching approach [197, 210], reducing the number of denoising steps during inference from 100 to 5.

To reduce training overhead, we implement a series of system-level optimizations for computational efficiency, such as faster token sampling, fewer camera inputs, optimized data loading, efficient attention implementation and mixed-precision training.

While none of these techniques is novel on its own, our contribution lies in carefully integrating them into a bimanual manipulation system.

3DFA achieves a new state of the art on the PerAct2 simulation benchmark, with a success rate of 85.1%. Furthermore, it outperforms strong baselines—including π_0 [15]—both in simulation and on a real-world 10-task benchmark we constructed using the bimanual ALOHA platform [84]. We conduct an extensive ablation study to break down the contributions of each design choice.

Notably, 3DFA is a general-purpose framework capable of predicting both sparse keyposes and dense end-effector trajectories, and is applicable to both unimanual and bimanual manipulation. On the 18 single-arm tasks of PerAct [312], 3DFA is trained to predict the next end-effector keypose and performs on par with 3DDA, while requiring significantly less training and inference time. Furthermore, on the 74-task benchmark of [102], 3DFA excels at jointly predicting the next keypose and the trajectory connecting it to the current pose in a single forward pass, outperforming the best baseline by 7.3%. These results highlight the versatility and efficiency of 3DFA as a framework for 3D manipulation.

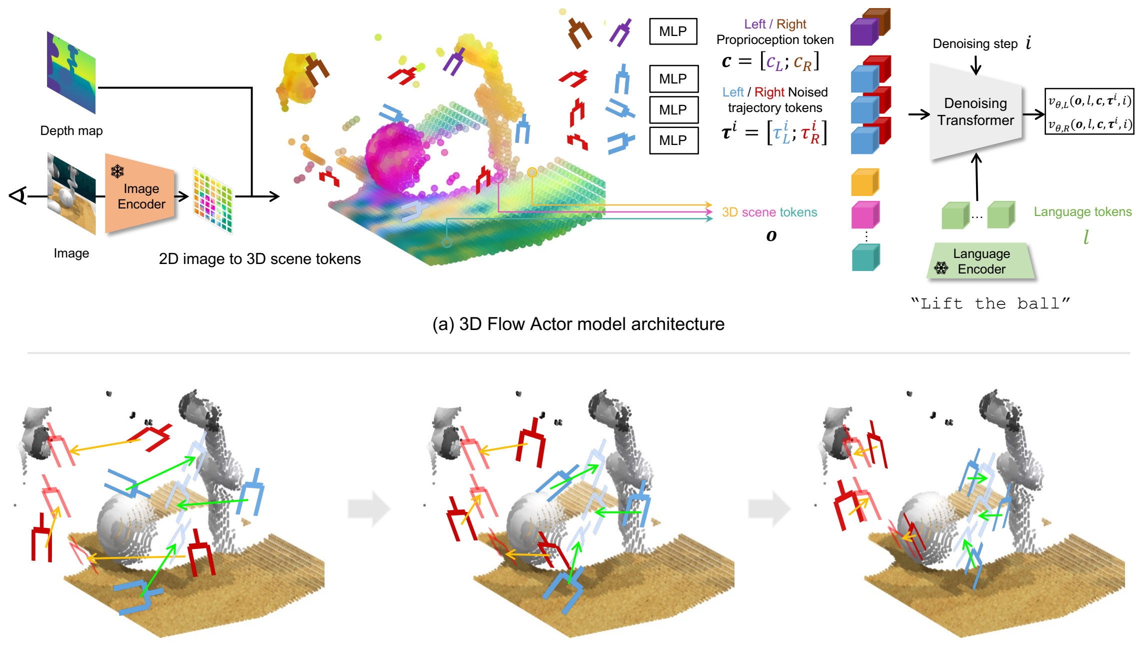
In summary, our contributions are: (1) Adaptation of state-of-the-art single-arm 3D generative policies to bimanual manipulation, (2) Dramatic acceleration of training and inference time in 3D policies, (3) State-of-the-art bimanual manipulation results on PerAct2, with an absolute margin of 53.1% over the previous state of the art, and 41.4% over π_0, (5) State-of-the-art bimanual manipulation results in the real world, outperforming foundational policies in a direct finetuning comparison, (6) Competitive single-arm manipulation results on the PerAct 18-task benchmark, and (7) State-of-the-art results on the HiveFormer unimanual 74-task benchmark, demonstrating trajectory prediction capabilities. Our models, code and videos of our manipulation results are available at https://3d-flow-actor.github.io/.

9.2 Related Work

Bi-manual manipulation. Bimanual manipulation is challenging due to the need for precise coordination between two arms. A key bottleneck is the difficulty of collecting large-scale, high-quality bimanual demonstration data, which has historically constrained the scalability of approaches [97, 98]. Recent works [84, 405] have introduced more cost-effective pipelines and platforms for scaling real-world data collection. However, these methods primarily rely on RGB inputs and struggle to generalize across diverse tasks, object types, scene configurations or robot embodiments. To address these limitations, several multi-task simulation benchmarks have been proposed to facilitate large-scale demonstration collection and policy evaluation. PerAct2 [100] extends the RLBench [139] benchmark to support multi-task bimanual manipulation, with expert demonstrations generated using sampling-based motion planners [176]. RoboTwin [247] introduces a generative digital twin framework, leveraging 3D generative foundation models and large language models to create diverse expert datasets and real-world-aligned evaluation environments.

In parallel, the development of bimanual manipulation policies generally falls into two main categories. One line of research extends single-arm policy architectures to bimanual. For example, VoxAct-B [200] builds upon the 3D voxel-based scene representations of [312] to jointly predict object and end-effector poses. Similarly, RDT-1B [204] follows the Vision-Language-Action (VLA) paradigm [15, 20, 275], fine-tuning large language models to improve generalization to diverse task instructions. The second line of research composes multiple single-arm policies into a unified bimanual policy [148, 220, 245]. For instance, DIF [148] observes that bimanual manipulation involves both independent and coordinated actions between the two arms, so it proposes to learn separate single-arm policies and a coordination module to combine them for coordinated tasks. Our work belongs to the first category and extends the action space of [164] to predict pose trajectories for both arms simultaneously.

Diffusion and Flow Matching models in robotics. Diffusion models have emerged as powerful tools in imitation learning [38, 93, 286] and offline RL [26, 106, 376]. DDPM [117] has been the most widely adopted diffusion algorithm to iteratively add and remove Gaussian noise to and from the samples, following conditional probability paths. More recently, Flow Matching [197, 210] has drawn attention in robot learning. Policies that use flow matching learn to predict the velocity field directly pointing toward the target action [13, 15, 18, 39, 59, 86, 287, 349, 398, 400], resulting in better sampling efficiency and lower computational cost than DDPM-based policies. 3DFA also incorporates Flow Matching, yielding a state-of-the-art bimanual manipulation policy with real-time inference capabilities.


Depth map
Image
Image Encoder
2D image to 3D scene tokens
MLP
MLP
MLP
MLP
Left / Right Proprioception token
$c = [c_L; c_R]$
Left / Right Noised trajectory tokens
$\tau^i = [\tau^i_L; \tau^i_R]$
3D scene tokens
o
Denoising step i
Denoising Transformer
$v_{\theta,L}(o, l, c, \tau^i, i)$
$v_{\theta,R}(o, l, c, \tau^i, i)$
Language tokens
l
Language Encoder
"Lift the ball"

(a) 3D Flow Actor model architecture

(b) Inference

Figure 9.1: **Method overview**. **Top:** 3DFA is a flow matching policy that builds atop a performant 3D diffusion policy–3DDA [164]. Our model encodes the visual scene **o**, left- and right-hand robot proprioception c_L, c_R, and left- and right-hand noised trajectory $\boldsymbol{\tau}^i_L, \boldsymbol{\tau}^i_R$ into 3D tokens. Taking as input language tokens l, diffusion step i, and these 3D tokens, 3DFA outputs the left- and right-hand velocity field $v_{\theta,L}, v_{\theta,R}$. **Bottom:** During inference, 3DFA iteratively predicts the straight-line velocity field pointing toward the left- and right-hand target pose.

9.3 Method

3D Flow Actor builds upon the state-of-the-art single-arm 3D diffusion policy, 3DDA [164]. To achieve faster inference and reduce training overhead, we replace the original DDPM-based diffusion mechanism with a more efficient flow matching approach and introduce a series of system-level optimizations. In this section, we begin by briefly reviewing the flow matching framework and the 3DDA architecture. We then detail our extensions to support bi-manual manipulation, along with the specific optimizations implemented to enhance performance and efficiency.

9.3.1 Flow Matching for Fast Action Generation

Diffusion models are powerful generative frameworks that excel in modeling multi-modal data distributions. Typical diffusion models based on DDPM [117] generate samples by iteratively removing Gaussian noise, following a stochastic process defined by conditional probability transitions. In contrast, flow matching approaches [197, 210] generate data by solving an optimal transport problem between a source distribution π_0 and a target distribution π_1.

In particular, we adopt Rectified Flow [210], an efficient instantiation of flow matching that transforms a sample $X_0 \sim \pi_0$ into a target sample $X_1 \sim \pi_1$ by following nearly straight paths in the sample space. This streamlined approach significantly reduces the computational overhead associated with inference while retaining the expressiveness needed for high-dimensional generative tasks, making it especially well suited for robotics, where real-time performance is critical.

The rectified flow between (X_0, X_1) defines a continuous, time-differentiable trajectory $\mathbf{Z} = Z_t : t \in [0,1]$ that transports X_0 to X_1 and is governed by the ordinary differential equation (ODE):

$$dZ_t = v_t^X(Z_t)dt, \quad t \in [0,1], \quad Z_0 = X_0, \tag{9.1}$$

where v_t^X is a time-dependent velocity field. The optimal velocity field that minimizes the expected discrepancy from the straight-line path between X_0 and X_1 can be formally defined as: $\inf_v \int_0^1 \mathbb{E}[\|X_1 - X_0 - v(X_t,t)\|]dt$, where $X_t = (1-t)X_0 + tX_1$ denotes the linear interpolation at time t [210]. The model is trained to approximate this velocity field by minimizing the loss [210]:

$$\mathcal{L}_\theta = \mathbb{E}_{t,X}[\|v_\theta(X_t,t) - v_t^X\|^2] \tag{9.2}$$

In our setting, the goal is to generate robot actions from noise. We define $\pi_0 \sim \mathcal{N}(0,I)$ as the prior distribution over Gaussian noise and π_1 as the distribution over real actions. During inference, the model iteratively transforms the noise sample X_0 into a predicted action X_1 over N steps with a fixed step size $\Delta t = \frac{1}{N}$: $X_{t+\Delta t} = X_t - \Delta t \cdot v_\theta(X_t,t)$.

9.3.2 3D Diffuser Actor

3DDA [164] is trained to imitate demonstration trajectories of the form of $\{(\mathbf{o}_1, \mathbf{a}_1), (\mathbf{o}_2, \mathbf{a}_2), ...\}$, accompanied with a task language instruction l, where $\mathbf{o}_t$ stands for the visual observation and $\mathbf{a}_t$ stands for robot action at timestep t. Each observation $\mathbf{o}_t$ is one or more posed RGB-D images. Each action $\mathbf{a}_t$ is a single-arm end-effector pose and is decomposed into 3D location, rotation and binary (open/close) state: $\mathbf{a}_t = \{\mathbf{a}_t^{\text{loc}} \in \mathbb{R}^3, \mathbf{a}_t^{\text{rot}} \in \mathbb{R}^6, \mathbf{a}_t^{\text{open}} \in \{0, 1\}\}$. Let $\boldsymbol{\tau}_t = (\mathbf{a}_{t:t+T}^{\text{loc}}, \mathbf{a}_{t:t+T}^{\text{rot}})$ denote the trajectory of 3D locations and rotations at timestep t, of temporal horizon T. 3DDA, at each timestep t predicts a trajectory $\boldsymbol{\tau}_t$ and binary states $\mathbf{a}_{t:t+T}^{\text{open}}$.

3DDA conditions on 3D scene feature representations derived from posed cameras and sensed depth and uses DDPM-based diffusion to predict the noise component at each diffusion step. To simplify the notation, we denote $\boldsymbol{\tau}^i$ as the noisy trajectory estimate at diffusion step i without specifying the trajectory timestep. The model conditions on the following inputs: (1) 3D scene tokens: 3DDA featurizes image views using a 2D image encoder and lifts each of the feature patches to 3D by calculating the average 3D location of each patch; (2) 3D proprioception tokens: 3DDA contextualizes a set of learnable embeddings with 3D scene tokens based on the proprioceptive location; (3) 3D trajectory tokens: 3DDA maps each noisy action $\mathbf{a}^i$ of trajectory $\boldsymbol{\tau}^i$ at diffusion step i to a latent feature vector and lifts these feature vectors to 3D, based on the noisy location estimate of $\mathbf{a}^i$; (4) language tokens: language instructions are encoded to latent embeddings with a text encoder. 3DDA fuses all 3D tokens using relative 3D attentions, and additionally fuses language tokens using normal attentions. As a last step, the refined trajectory tokens are fed to MLPs to predict the noise added to $\boldsymbol{\tau}^0$, as well as the end-effector opening. We refer to [164] for more details.

9.3.3 3D Flow Actor

To extend 3D Diffuser Actor to bi-manual manipulation, we first redefine the robot action in a bi-manual form: $\mathbf{a}_{t,L}$ and $\mathbf{a}_{t,R}$ denote the robot action at timestep t, of the left and right robot arm respectively. Our goal is to predict the corresponding trajectory $\boldsymbol{\tau}_{t,L} = (\mathbf{a}_{t:t+T,L}^{\text{loc}}, \mathbf{a}_{t:t+T,L}^{\text{rot}})$ and $\boldsymbol{\tau}_{t,R} = (\mathbf{a}_{t:t+T,R}^{\text{loc}}, \mathbf{a}_{t:t+T,r}^{\text{rot}})$ of temporal horizon T for both arms. Obviously, to apply 3DFA on unimanual setups, we still use the unimanual trajectory definition: $\mathbf{a}_t$ and $\boldsymbol{\tau}_t = (\mathbf{a}^{\text{loc}}_{t:t+T}, \mathbf{a}^{\text{rot}}_{t:t+T})$.

We follow the same 3D tokenization procedure to map (1) the noisy estimate of pose $\mathbf{a}^i_L$ of $\boldsymbol{\tau}^i_L$ and $\mathbf{a}^i_R$ of $\boldsymbol{\tau}^i_R$ at denoising step i, and (2) the left- and right-hand proprioceptive information c_L and c_R, into 3D tokens. We use the same 3D Relative Denoising Transformer architecture to contextualize all these tokens and predict the translation and rotation noise and the end-effector opening for both arms.

We replace the DDPM-based diffusion method with *rectified flow*. The noisy left- and right-hand trajectory estimate $\boldsymbol{\tau}^i_L = (1-i)\epsilon_L + i\boldsymbol{\tau}^0_{t,L}$ and $\boldsymbol{\tau}^i_R = (1-i)\epsilon_R + i\boldsymbol{\tau}^0_{t,R}$ become the linear interpolation at denoising step i, where $\boldsymbol{\tau}^0_{t,L}$ and $\boldsymbol{\tau}^0_{t,R}$ denote the clean trajectory, and ϵ_L and ϵ_R denote the sampled noise for the left- and right-hand end effector. In particular, our model takes as input two-hand trajectory estimate tokens $\boldsymbol{\tau}^i = \{\boldsymbol{\tau}^i_L, \boldsymbol{\tau}^i_R\}$, proprioception tokens $\mathbf{c} = \{c_L, c_R\}$, language tokens l, and scene tokens $\mathbf{o}$; it outputs the left- and right-hand velocity field $v_{\theta,L}, v_{\theta,L}$ and gripper openess $f^{\text{open}}_{\theta,L}, f^{\text{open}}_{\theta,L}$, respectively. We ignore time step t to simplify notations.

During training, we sample a time step t uniformly, denoising step $i \sim \sigma(\mathcal{N}(0, I))$ from a logit-normal distribution and noise $\epsilon_L \sim \mathcal{N}(0, I), \epsilon_R \sim \mathcal{N}(0, I)$ from Gaussian distribution. We use the $L2$ loss to supervise the velocity field and binary cross-entropy (BCE) to supervise the end-effector opening. By ignoring the notation of time step t, the total objective reads:

$$\begin{aligned}\mathcal{L}_\theta = {} & \|\epsilon_L - \boldsymbol{\tau}^0_L - v_{\theta,L}(\mathbf{o}, l, \mathbf{c}, \boldsymbol{\tau}^i, i)\|^2 + \|\epsilon_R - \boldsymbol{\tau}^0_R - v_{\theta,R}(\mathbf{o}, l, \mathbf{c}, \boldsymbol{\tau}^i, i)\|^2 \\ & + \text{BCE}(f^{\text{open}}_{\theta,L}(\mathbf{o}, l, \mathbf{c}, \boldsymbol{\tau}^i, i), \mathbf{a}^{\text{open}}_{1:T,L}) + \text{BCE}(f^{\text{open}}_{\theta,R}(\mathbf{o}, l, \mathbf{c}, \boldsymbol{\tau}^i, i), \mathbf{a}^{\text{open}}_{1:T,R}),\end{aligned} \tag{9.3}$$

System-level optimizations We identify and improve 3DDA's implementation bottlenecks, both in the model and in the training pipeline. First, we reduce the number of cameras needed and only use one static camera and wrist cameras, a setup that is both practical and boosts performance over single-camera setups, as we show in our ablations. This significantly saves computation from the visual backbone and leads to fewer scene tokens, thus more efficient attention.

Furthermore, we adopt density-biased sampling (DBS) [257] to replace farthest point sampling (FPS) [73]. In more detail, 3DDA employs FPS in the feature space to sparsify the scene tokens. FPS maintains a set of candidate points and a set of sampled points. Then, it iteratively samples the candidate point with maximum average distance from all sampled points. We replace this with DBS, which first estimates the sparsity of a neighborhood around a point as the average distance of the $k = 8$ nearest neighbors of that point. Then, it promotes sampling in the sparser neighborhoods. A fast batched version of DBS can be implemented in pure PyTorch [260].

Lastly, by carefully engineering the data loader, mixed-precision training and efficient attention implementation, we speed up training significantly. We refer to Section H.2 for more details.

9.4 Experiments

9.4.1 Evaluation on PerAct2

PerAct2 [100] is a simulation benchmark for bimanual manipulation, equipped with two Franka Panda robots. Methods are trained to predict the next end-effector keypose [100] and then an RRT-based motion planner [176] generates a trajectory of joint angles connecting the current to the predicted pose. PerAct2 has a suite of 13 bimanual tasks, each of which has 1-5 variations that concern object poses, appearance and semantics (more in Section H.3.1). Each task is accompanied by 100 expert demonstrations for training and 100 episodes for evaluation. There are five calibrated RGB-D cameras available, front, left wrist, right wrist, left shoulder, and right shoulder, which capture images at 256×256 resolution. Notably, 3DFA does not need to use all five cameras but only the front, left wrist, and right wrist cameras. In contrast, other baselines, unless otherwise stated, use all five cameras. We adopt the task completion success rate as our evaluation metric.

Baselines We compare against: ACT [405], RVT-LF [95, 100] and PerAct-LF [100, 312], PerAct^2 [100] (results copied from [100]); AnyBimanual [220]; 3D Diffusion Policy (DP3) [393] and KStar Diffuser [224] (results from [224]); PPI [378]; π_0 [15]. Notably π_0 is a 2D robot generalist pre-trained on 10,000 hours of robot data and capable of performing bimanual manipulation. We adapt it to predict end-effector keyposes and fine-tune it using three cameras. See Section H.3.6 for more details on the baselines.

We identify different settings among our model and baselines: (1) our method,

	multi-task training	Avg. Success	push box	lift ball	push buttons	pick up plate	put item into drawer	put bottle into fridge
ACT [405]	✗	5.9	0	36	4	0	13	0
RVT-LF [95, 100]	✗	10.5	52	17	39	3	10	0
PerAct-LF [100, 312]	✗	17.5	57	40	10	2	27	0
PerAct^2 [100]	✗	16.8	6	50	47	4	10	3
DP3 [393]	✗	-	56	64	-	-	-	-
KStar Diffuser [224]	✗	-	83	98.7	-	-	-	-
PPI [378]	✗	-	**96.7**	89.3	-	-	79.7	-
AnyBimanual [220]	✓	32	46	36	73	8	-	26
π_0 [15]	✓	43.7	93	97	38	41	40	22
3DFA (ours)	✓	**85.1**	$92.7_{\pm 0.47}$	$\mathbf{99.7}_{\pm 0.47}$	$\mathbf{92.7}_{\pm 1.89}$	$\mathbf{69.7}_{\pm 12.6}$	$\mathbf{93.0}_{\pm 2.83}$	$\mathbf{89.3}_{\pm 1.89}$

	multi-task training	handover item	pick up laptop	straighten rope	sweep dust	lift tray	handover item (easy)	take tray out of oven
ACT [405]	✗	0	0	16	0	6	0	2
RVT-LF [95, 100]	✗	0	3	3	0	6	0	3
PerAct-LF [100, 312]	✗	0	11	21	28	14	9	8
PerAct^2 [100]	✗	11	12	24	0	1	41	9
DP3 [393]	✗	-	6.3	-	1.7	-	0	-
KStar Diffuser [224]	✗	-	43.7	-	89	-	27	-
PPI [378]	✗	-	46.3	-	98.7	92	62.7	-
AnyBimanual [220]	✓	15	7	24	67	14	44	24
π_0 [15]	✓	2	27	7	2	72	59	68
3DFA (ours)	✓	$\mathbf{89.0}_{\pm 7.12}$	$\mathbf{74.0}_{\pm 8.96}$	$\mathbf{40.7}_{\pm 1.89}$	$\mathbf{99.3}_{\pm 0.47}$	$\mathbf{94.7}_{\pm 0.47}$	$\mathbf{96.0}_{\pm 5.65}$	$\mathbf{94.7}_{\pm 1.89}$

Table 9.1: **Evaluation on PerAct2** (3 random seeds). 3DFA,[220],[15] are multi-task trained and evaluate the final checkpoint across all tasks, while others are single-task trained and report the best checkpoint per task. **3DFA outperforms all prior arts on most tasks by a large margin.**

π_0 and AnyBimanual adopt multi-task learning and test the final checkpoint, while others train single-task models and select the best intermediate checkpoint per task for evaluation; (2) AnyBimanual, DP3, PPI and KStar Diffuser do not test on all, but a subset of 13 tasks. Our settings are more general and practical, as multi-task learning on all tasks is essential for building a robot generalist [15, 20].

Results We show quantitative results in Table 9.1. 3DFA largely outperforms all baselines, with an absolute improvement of 68.3% over PerAct^2. When isolating the same 5 tasks in which both PPI and KStarDiffuser report results, 3DFA achieves 92.3%, outperforming them by 13.6% and 24% absolute respectively. Notably, our method with 3.8M parameters outperforms π_0, with nearly 1000 times more parameters. We show that, although large-scale pretraining is useful, explicitly incorporating 3D information into the model is a strong inductive bias. We also test 3DDA on this benchmark, which also achieves an average success rate of 85%. We discuss failure

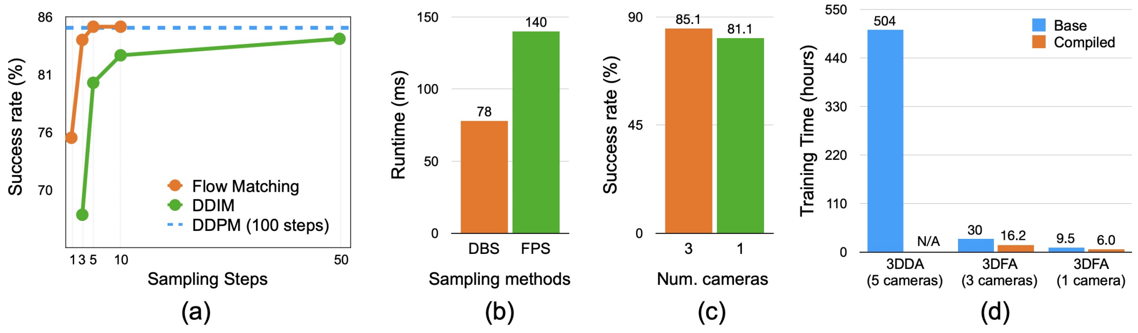


Figure 9.2: **Ablation study.** on PerAct2. Left to right we compare: (a) the success of flow matching against DDIM and DDPM with varying numbers of denoising steps, (b) the runtime of density-based sampling against farthest point sampling, (c) the success rates under different numbers of cameras and (d) the training time of 3DFA and 3DDA. Our design choices boost effectiveness and efficiency.

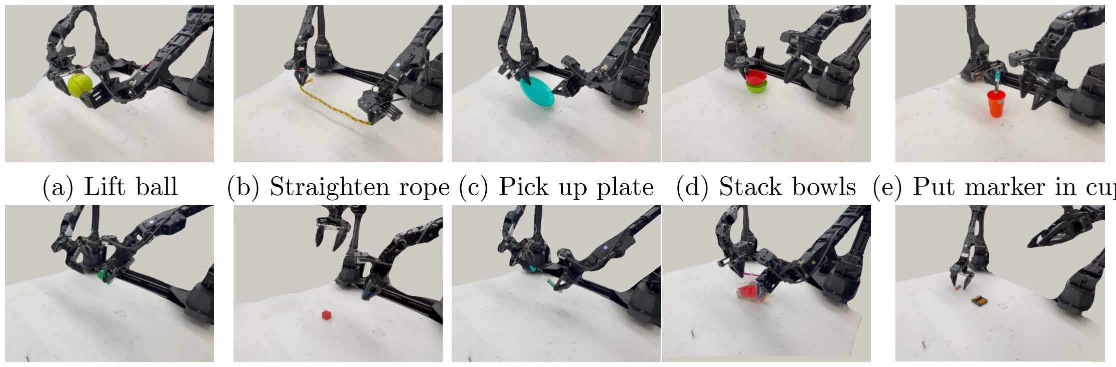
cases in Section H.3.10.

Ablations We first compare the different denoising methods in terms of performance and inference time in Figure 9.2a. We observe that i) DDPM is the least flexible, requires 100 denoising steps and runs at 0.74Hz while achieving 85.0% SR; ii) we apply DDIM [322] to reduce the denoising steps. While the runtime improves (13.3Hz at 5 steps), the performance gradually drops; iii) flow matching is the most robust, maintaining all the performance even when running at 5 steps and 13.3Hz, nearly all the performance at 3 steps and 20Hz, and 75.2% in success rate at 1 step and 26Hz.

Next, we ablate the effect of the point sampling strategy in Figure 9.2b. While both versions achieve the same success rate, a forward pass for 3DFA with DBS costs 75ms, while for FPS it costs 140ms.

Third, we test the impact of wrist cameras in Figure 9.2c. We implement a single-camera version of 3DFA that uses only the front camera, with all other hyperparameters fixed. This model achieves 81.1% and runs at 66.7Hz. Although in many tasks the performance is similar to the 3-camera 3DFA, there are tasks such as "push buttons" where the gap is significant, due to the importance of first-person observations for seeing and reaching the relevant objects.

Lastly, in Figure 9.2d, we compare the training time of 3DFA to that of the original implementation of 3DDA when trivially adapted for bimanual manipulation. Both trained on 4 A100 GPUs, 3DDA takes 21 days, 3DFA only needs 30 hours. Notably, the training time can be significantly reduced when combined with CUDA



(a) Lift ball (b) Straighten rope (c) Pick up plate (d) Stack bowls (e) Put marker in cup

(f) Hand over block (g) Stack blocks (h) Open marker (i) Close ziploc (j) Insert battery

Figure 9.3: **Real-world benchmark.** Our real-world benchmark has a suite of 10 bimanual tasks, including 5 easy tasks (top row) and 5 difficult tasks (bottom row).

graph compilation (`torch.compile`) or using fewer cameras, options not applicable to the original 3DDA implementation.

9.4.2 Evaluation in the real world

We construct a challenging real-world multi-task manipulation benchmark using a two-handed robot system, Mobile Aloha [83], equipped with a front-facing ZEDX RGB-D sensor and two wrist-mounted RealSense D405 RGB-D sensors. Our benchmark (Figure 9.3, Section H.3.2) consists of 10 tasks that often require more precise and synergistic two-hand action than those of PerAct2, such as *open marker*, *close ziploc* and *insert battery*. We teleoperate Mobile Aloha to collect 40 demonstrations per task for training, capturing images and actions at a frequency of 5 Hz. All models are fine-tuned/trained on the same set of demonstrations and then tested on 20 episodes per task.

Baselines We compare against: (1) π_0 [15]; (2) iDP3 [392], a variant of DP3 with improved architectures, carefully designed for two-hand humanoids. Both our model and the baselines are trained for closed-loop trajectory prediction, without keyposes. Following the original papers, both baselines predict the robot's joint angles, rather than end-effector poses that our model predicts. To mimic human operation, we train both models to output joint values of the ALOHA leader arms, and the ALOHA follower arms are tasked to follow the action. In contrast, our 3DFA predicts end-effector poses of the ALOHA follower arms, which are transformed into joint angles

	Avg. Success	Inf. speed	Params in M	lift ball	straighten rope	pick up plate	stack bowl	put marker into bowl	handover block	stack blocks	open marker	close ziploc	insert battery
π_0	32.5	100ms	3238	85	75	40	20	20	15	35	20	10	5
iDP3†	24.5	420ms	68.8	45	35	30	40	35	25	15	10	10	0
3DFA (ours)	**53.5**	54ms	3.8	**85**	**75**	**80**	**80**	**70**	**60**	**45**	**25**	**10**	**5**

Table 9.2: **Evaluation in the real world.** 3DFA and π_0 adopt multi-task learning settings, while iDP3 uses a single-task learning setup. **3DFA outperforms π_0 and iDP3 on all tasks by a large margin**.

by solving inverse kinematics. 3DFA and iDP3 are 3D policies and require accurate depth information. However, the depth maps of the wrist cameras are very noisy, which hurts performance. We do not use wrist cameras as input to both models. On the other hand, 2D policies like π_0 have no such limitations and take as input all three cameras. The input images are downsampled to a resolution of 256×256 during training and inference.

Results We show the quantitative results in Table 9.2. 3DFA outperforms π_0 and iDP3 on most tasks. We find that π_0 is sensitive to occlusion and cannot locate and reach the object if it is not visible in the wrist observations. We also find that iDP3 struggles to predict precise end-effector poses, as the robot often approaches but fails to grasp the object. In contrast, our method consistently outperforms these baselines in most tasks. We refer to Sections H.3.9, H.3.10 for more analysis.

9.4.3 3DFA as a unimanual policy: a more accessible drop-in replacement for 3DDA

To highlight the versatility of 3DFA and the broader application of our technical contributions beyond bimanual manipulation, we additionally train and test 3DFA as a unimanual policy.

We first use the **18-task PerAct benchmark** [312] to allow a direct comparison with 3DDA [164] in terms of performance and efficiency. Policies are trained in a multi-task fashion on 100 demos per task and tested on 25 episodes per task. There are four calibrated RGB-D cameras available, front, wrist, left shoulder, right shoulder. More details in Section H.3.3.

We compare two variants of 3DFA and 3DDA in Figure 9.4. All models are trained for keypose prediction. 3DFA with four cameras achieves competitive performance to

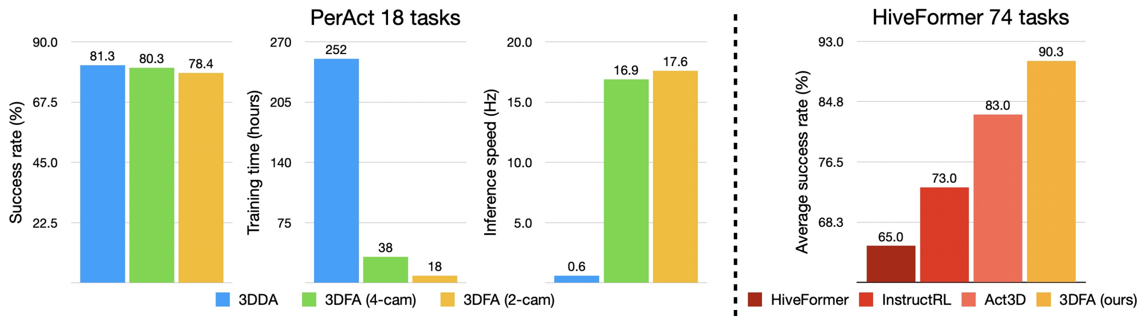


Figure 9.4: **Single-arm manipulation results. Left**: 3DFA performs on par with 3DDA on the PerAct 18-task benchmark, while requiring less than 1/6th of 3DDA's training time and being 28 times faster during inference. **Right**: 3DFA achieves a new state-of-the-art on the 74 tasks of HiveFormer.

	Avg. Success	Num. stages	unplug charger	close door	open box	open fridge	frame off hanger	open oven	books in shelf	shoes in box
DP3	28.5	1	33.3	76.0	98.3	4.3	12.3	0.3	3.7	0.0
PointFlowMatch	67.8	1	83.6	68.3	99.4	31.9	38.6	75.9	68.8	76.0
ChainedDiffuser	84.5	2	95.0	76.0	96.0	68.0	85.0	86.0	92.0	**78.0**
3DFA (ours)	**91.3**	1	**99.0**	**96.0**	**100.0**	**70.0**	**96.0**	**99.0**	**99.0**	71.0

Table 9.3: **Evaluation on 8 RLBench tasks** from [364] that require continuous interaction with the environment. 3DFA predicts dense trajectories and outperforms all baselines by a large margin.

3DDA, while running 28 times faster and requiring 6 times less training time. When using two cameras only (front and wrist), 3DFA reduces training and inference time even further, 14 and 30 times over 3DDA, with a small performance drop. Detailed results in Section H.3.7.

We then employ the **74-task benchmark of HiveFormer** [102] to demonstrate the ability of 3DFA to predict dense trajectories instead of sparse keyposes. Policies are trained on one task at a time, using 100 demos, and tested on 100 episodes. We compare our performance on all 74 tasks and also separately consider a subset of 8 challenging tasks [39, 364] that require continuous interaction with the environment and thus cannot be solved by simple predicting keyposes- more in Sections H.3.4, H.3.5.

We compare against: (1) keypose-prediction models HiveFormer [102], InstructRL [199] and Act3D [91], which use three cameras; (2) close-loop trajectory-prediction models PointFlowMatch [39] and DP3 [393], which use five cameras; (3) Chained-Diffuser [364], a two-stage policy consisting of a keypose predictor and a keypose-conditioned trajectory predictor, using three cameras.

3DFA is trained to jointly predict the next end-effector keypose and the dense trajectory until the next keypose in a non-hierarchical, single-forward-pass fashion. It relies on two camera observations, front and wrist, to maintain consistency with the rest of our experiments. More details in Section H.3.6.

We show results on all 74 tasks in Figure 9.4. 3DFA achieves a new state of the art of 90.3%, improving over Act3D by 7.3%. We compare on 8 challenging tasks in Table 9.3. Our method outperforms the two-stage ChainedDiffuser by 6.8% and PointFlowMatch by 23.5% absolute, highlighting its ability to predict continuous trajectories. Detailed breakdown of performance on all tasks in Section H.3.8.

9.5 Conclusion

We present 3D Flow Actor, a policy that extends 3D Diffuser Actor to bimanual manipulation. Our method sets a new state of the art on both a simulation benchmark-PerAct2-and a real-world benchmark by a large margin, outperforming much larger foundational policies. Reducing camera requirements and training/inference times make 3DFA much more applicable to real-world applications than previous 3D generative policies. We demonstrate that our design choices are also effective in single-arm manipulation, where 3DFA exhibits vast training and inference speedups on PerAct, together with a new state-of-the-art on 74 RLBench tasks, including challenging tasks that require dense trajectory prediction. Our work is a clear demonstration that even in the era of large-scale VLAs, policies that make use of 3D scene representations are still relevant. An exciting future direction is to scale up our policy by training on large sets of real-world data and to further enhance spatial and dynamic reasoning for contact-rich tasks. As concurrent work in the vision community continues to increase the amount of well-calibrated robotics data, the improvements introduced in this paper will only become more necessary.

Chapter 10

Conclusion and Future Work: Scaling 3D Policies to Billions of Parameters

10.1 So Far

This thesis explored how unifying 3D spatial representations with generative models can serve as a foundational strategy for building generalist robot policies. We began by addressing fundamental challenges in 3D perception, developing models that bridge 2D and 3D visual understanding. Through open-vocabulary detection, joint 2D-3D segmentation, and memory-prompted point cloud reasoning, we established critical perceptual bottlenecks that enable flexible, cross-domain task generalization. These contributions laid the architectural groundwork for end-to-end 3D policy learning, where perception is not an isolated pre-processing step but an integral part of the control loop.

Building on these perceptual foundations, we introduced a family of generative 3D policies that can model the complex, multimodal nature of robotic manipulation. From goal-conditioned planners to full trajectory diffusion models, we demonstrated how explicitly grounding policies in 3D space enables robust reasoning over diverse tasks and embodiments. The development of 3D Flow Actor (3DFA) highlighted how an efficient architectural design can outperform significantly larger 2D models in both performance and computational efficiency.

10.2 Scaling 3D Policies

One natural question would be whether the advantages of 3D policies persist when we scale the model capacity and data, or the 3D inductive biases are an obstacle to successful scaling. Indeed, the advances on 3D policy learning often hinge on the availability of accurate and calibrated RGB-D sensors. While not impractical, this requirement limits the scale of data 3D policies can leverage. Large-scale datasets frequently lack depth observations or provide only noisy depth maps with inaccurate calibration, constraining efforts to scale 3D policies. In contrast, 2D policies sidestep these calibration requirements and can be trained on diverse datasets mapping images to actions. This flexibility has enabled the rise of generalist policies trained on large real-world datasets, often powered by vision-language models (VLMs) for their rich visual and linguistic understanding [15, 275, 287]. Can we scale 3D policies to the same extent as their 2D counterparts while preserving their structural advantages?

We redesign 3DFA by incorporating VLM representations and significantly expanding the capacity of its action decoder through deeper attention layers. The resulting model, 3DFA-VLA, scales to a billion of parameters (300x more parameters than 3DFA), based on three key modifications:

1. Vision-Language Encoder Upgrade: We replace CLIP [276] with Florence2 [366] as the vision-text encoder and allow it to be fine-tuned. This substitution alone increases the model size by nearly 500 million parameters, providing richer visual-textual representations.
2. Cross-Attention in Action Decoder: We redesign the attention mechanism in the action decoder, replacing its self-attention layers with cross-attention layers. This change restricts updates to the trajectory tokens while keeping the scene tokens frozen, drastically reducing the memory footprint. This modification does not increase the parameter count, but unlocks capacity for a much larger decoder.
3. Scaling the Action Decoder: Leveraging the memory savings, we significantly

scale the action decoder by increasing the number of attention layers from 6 to 24 and expanding the latent dimension from 120 to 1024. This contributes an additional 500 million parameters, enabling more expressive action representations.

While certain design choices, such as the decoder depth and the use of Florence2, are inspired by FLOWER-VLA [287], 3DFA-VLA remains a principled scaling of 3DFA and retains its coding structure.

10.2.1 Evaluation on CALVIN

We evaluate 3DFA-VLA on CALVIN [236], a language-conditioned manipulation benchmark featuring four environments (A, B, C, D) that vary in texture and object arrangements. Following the zero-shot protocol, we train on environments A, B, and C (17,000 episodes) and test on D (1000 episodes), measuring success by the average length of correctly executed task sequences (5 tasks per sequence, needed to be executed in order).

Unlike RLBench, where data scarcity is a bottleneck, CALVIN provides an order of magnitude more episodes, making it a potential testbed for scaling policies in simulation. However, CALVIN's main axis of difficulty is generalization to texture and has favored 2D models that leverage large VLMs [287] or video prediction methods [121, 332]. Among 3D approaches, only 3D Diffuser Actor (3DDA) has shown competitive results, attributed to its use of 2D visual features and generative diffusion objectives that handle noisy, multimodal distributions.

A critical observation here is the trade-off introduced by keypose-horizoned trajectory prediction. While chunking and interpolating trajectories into fixed-length segments (from keypose to keypose) simplifies the action space and enables training smaller models, it also reduces data diversity by an order of magnitude and introduces artifacts from forced interpolation. Moreover, due to the noisy demonstrations on CALVIN, the heuristics for keypose discovery may not be semantically meaningful, as in RLBench where there are clean motion patterns. In fact, we visualized the frames corresponding to keyposes on CALVIN, discovered using the algorithm we proposed in [164], and they often cannot be visually distinguished from arbitrary non-keypose frames. As a result, CALVIN's richness in observation-action pairs (approximately

900k) becomes underutilized when trajectories are abstracted into sparse keyposes (nearly one tenth of total annotations), limiting the effective dataset size for large models.

To address this, we hypothesize that closed-loop trajectory prediction, i.e., modeling the entire trajectory as a continuous sequence, enables better utilization of CALVIN's data and alleviates interpolation artifacts. We perform an ablation to validate this hypothesis.

We compare the following models: i) 3DFA trained for closed-loop trajectory prediction; ii) 3DFA trained with keypose-horizoned trajectories; iii) FLOWER-VLA [287], the current state-of-the-art 2D policy on CALVIN, which also uses Florence2 and similar architectural hyperparameters to 3DFA-VLA; iv) 3DFA-VLA trained with keypose-horizoned trajectories; v) 3DFA-VLA trained for closed-loop trajectory prediction. We also include [121, 164, 332] in the comparisons for reference.

Key findings (Figure 10.1):

- 3DFA-VLA (closed-loop) sets a new state-of-the-art on CALVIN, demonstrating that 3D policies scale with data and model capacity.
- 3DFA-VLA (keypose-horizoned) achieves strong performance but underperforms compared to its closed-loop counterpart, confirming that non-semantic trajectory abstraction may hinder scalability.
- The original 3DFA performs competitively with 3DDA under keypose-horizon supervision but lacks capacity to model closed-loop trajectories effectively and significantly underperforms larger models.
- 3DFA (closed-loop) barely solves any task, indicating that scaling the model capacity (as in 3DFA-VLA) is essential for closed-loop modeling on CALVIN.

10.2.2 Evaluation on PerAct2

We further evaluate on PerAct2 [100], a challenging bimanual manipulation benchmark featuring two Franka Panda arms. Unlike CALVIN, PerAct2 tasks emphasize precision and 3D reasoning over texture generalization, making it a stringent test for models relying on low-resolution visual representations, as we showed in the previous chapter with finetuning π_0 [15].

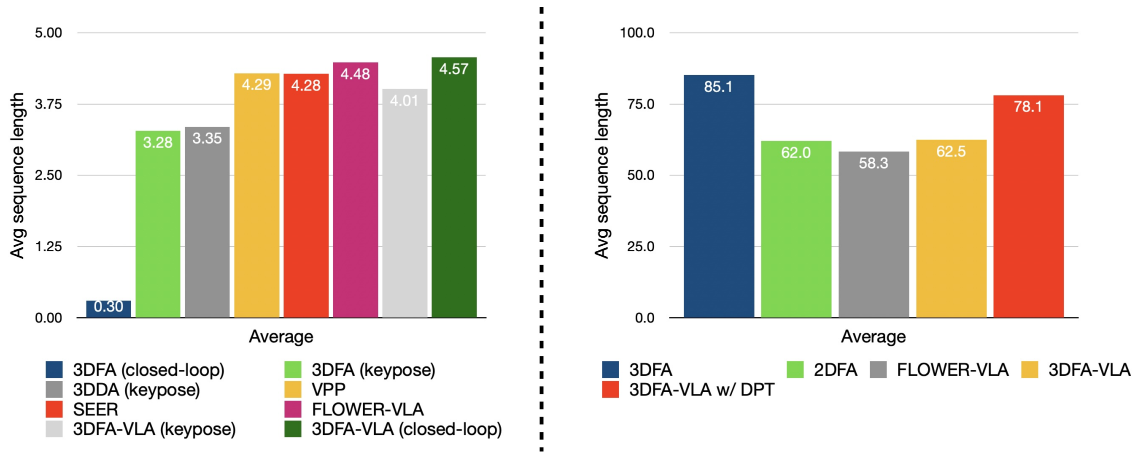


Figure 10.1: Left: 3DFA-VLA sets a new state-of-the-art on CALVIN. Right: Upsampling is the most crucial factor for generalization on precise RLBench tasks. Both results highlight the scalability of 3D-VLAs and their stronger performance over their 2D counterparts.

A notable architectural shift from CLIP to Florence2 introduces challenges here. Florence2, being a transformer, outputs fixed low-resolution features, unlike CLIP's progressively downsampled feature pyramid. Since our framework aligns 3D positional tokens to the spatial grid of visual features, this results in a significant loss of spatial granularity, which is detrimental for fine-grained 3D tasks like those in PerAct2.

To mitigate this, we integrate learnable upsampling layers from DPT [282] to enhance the spatial resolution of Florence2's feature maps, restoring the fine-grained localization necessary for precise manipulation.

We compare the following models: (i) 3DFA, as the SOTA reference; (ii) 2DFA, a model that uses identical architecture to 3DFA except for replacing the 3D attention with standard 2D attention; (iii) FLOWER-VLA; (iv) vanilla 3DFA-VLA; (v) 3DFA-VLA w/ DPT, which is 3DFA-VLA with the upsampling method from [282].

We show results in Figure 10.1. We draw the following conclusions:

- FLOWER-VLA, vanilla 3DFA-VLA and 2DFA all achieve statistically similar performance, indicating that the additional capacity does not make a difference on RLBench, but also that using 3D positions of very low resolution regresses the model to 2D behavior.
- 3DFA-VLA w/ DPT shows a large performance boost over its vanilla counterpart,

significantly (but not fully) closing the gap between 3DFA-VLA and 3DFA.

10.3 Conclusion and Open Directions

As we strive toward building truly generalist robots, it becomes increasingly clear that scaling data and model size alone is insufficient. Lessons from vision and language domains suggest that data-efficient architectures and learning objectives will be crucial, especially as we approach the natural limits of internet-scale textual data [269]. For robots, the challenge is even more pronounced: they must perceive, reason, and act in a world that is inherently 3D, dynamic, and multimodal.

The future of robot intelligence, therefore, lies at the intersection of large-scale learning and physically grounded representations. In this paradigm, policies must go beyond mimicking demonstrated behaviors; they must understand, adapt, and compose behaviors across tasks, environments, and embodiments. This thesis takes a step toward that vision, presenting a set of insights, models, and tools designed to guide the development of spatially-aware and efficient robot generalists.

At the same time, these insights open up new and exciting research avenues. Future work could achieve significant breakthroughs by pursuing directions such as:

1. Scaling 3D Data Acquisition in Simulation: As simulators become increasingly photorealistic, generating large-scale robot action datasets with clean depth and precise camera calibration will become much easier. This synthetic data can be used to train VLAs that build upon Vision-Language Models (VLMs), leveraging real-world passive data while bootstrapping from high-quality synthetic interactions.

2. Self-Supervised Auto-Calibration in the Real World: To accelerate real-world data collection, self-supervised computer vision techniques can be leveraged to automatically estimate metric depth and camera parameters, effectively "3D-ifying" 2D videos without manual calibration, lowering the barrier for scalable robot learning.

3. Hybrid 2D-3D Architectures for Universal Robot Policies: Developing architectures that seamlessly operate across 2D and 3D representations could enable training a single model on all available robot data. This could be similar in

fashion to our work ODIN.

4. Spatiotemporal Visual Memory for Long-Horizon Tasks: all policies we presented in this thesis rely on the current observation for decision-making, limiting their ability to reason about objects that have moved out of view. However, naively aggregating all past observations is computationally prohibitive. A promising direction is spatiotemporal point subsampling, inspired by the density-biased sampling in 3DFA, to retain only the most informative visual history..
5. Catastrophic Forgetting Mitigation via Multi-Task Co-Training: VLAs often fine-tune VLMs on action-only datasets, leading to catastrophic forgetting of their general visual-language understanding. Co-training policies simultaneously on action prediction and vision-language tasks could preserve the VLM's broad knowledge while enhancing action grounding. What are the right objectives that optimize for the resources required is an important question.
6. Factorized Policies for Robustness and Compositionality: End-to-end action prediction models risk overfitting to memorized motion patterns. An alternative is to factorize policies into subgoal predictors and low-level action planners, where subgoals are expressed in a generalizable representation. This could enhance robustness, enabling policies to recover from failures by recomposing behaviors dynamically. Choosing the right way to communicate subgoals to action planners while maintaining generality and scalability is the challenge to be addressed.
7. Beyond Imitation Learning: Fine-Tuning with Reinforcement and Reward Modeling: All policies in this thesis are trained via imitation learning, aligning with the distribution of demonstrations rather that of successful outcomes. Future work could explore fine-tuning pre-trained policies with reinforcement learning to close this gap. In parallel, learning reward functions from passive data or demonstrations remains an essential complementary challenge.

Ultimately, the path to generalist robot intelligence demands a shift from brute-force scaling to deliberate design of representations, objectives, and learning processes. This thesis provides a framework for that shift, offering models and insights that can guide the field toward more capable, efficient, and generalizable robot policies.

Appendix A

Bottom Up Top Down Detection Transformers for Language Grounding in Images and Point Clouds

A.1 Implementation details

We report here architecture choices as well as training hyperparameters. We implement BUTD-DETR in PyTorch. For the 3D version, the point cloud is encoded with PointNet++ [270] using the same hyperparameters as in [216], pre-trained on ScanNet [49]. We use the last layer's features, resulting in 1024 visual tokens. In the cross-modality encoder, instead of allowing the visual features to attend to the box features, we directly concatenated the box features to the input point cloud. Specifically, for all the points that lie inside a box, we concatenate this box's features directly to their point features (xyz and color). If a point lies inside multiple boxes, we randomly sample one box's features. Points that do not lie inside inside any box are padded with zeros. This is computationally cheaper than cross-attending visual features to box features and works well in 3D since the objects do not intersect. In 2D, however, it does not work well since the objects and thus their boxes overlap a lot and hence usually a pixel falls inside multiple boxes. We ablate more on this fusion in A.3. In the decoder, the queries are formed from the top 256 most confident visual tokens. To compute this confidence score, each visual token is fed to an MLP to give a scalar value. We supervise these values using Focal Loss [196]. Specifically, since each visual token corresponds to a point with known coordinates, we associate visual tokens to ground-truth object centers and keep the 4 closest points to each center. We consider these matched points as positives, i.e. here points with high ground-truth objectness. The same scoring method is employed in [216]. We set $N_E = 3$ with no self-attention layers, $N_D = 6$. All attention layers are implemented using standard self-/cross-attention [222, 343].

For the 2D version, the image is encoded using ResNet-101 [110] pretrained on ImageNet [55]. We use multi-scale features as in [412]. The feature maps of the different scales are flattened and concatenated in the spatial dimension, leading to 17821 visual tokens. The feature dimension of each token is 256. To obtain the box proposals, we use the detector of [7] trained on 1601 classes of Visual Genome [175]. The detected boxes are encoded using their spatial and categorical features. Specifically, we compute the 2D Fourier features of each box and feed them to an MLP, then we concatenate this vector with a learnable semantic class embedding and feed to another MLP to obtain the box embeddings. To form queries, we rank visual tokens based on their confidence score and keep the 300 most confidence ones. This confidence layer is supervised using Focal Loss [196]: we assign a positive objectness scores to every point that lies inside a ground-truth answer box. We set $N_E = 6$ and $N_D = 6$. All attention layers to the visual stream are implemented with deformable attention [412], attention to either the language stream or detected boxes is the standard attention of [222, 343].

For the 3D model, we use a learning rate of $1e{-}5$ for RoBERTa and $1e{-}4$ for all other layers. We are able to fit a batch size of 6 on a single GPU of 12GB. Under these conditions, each epoch takes around 3 hours. For the 2D model, we use a learning rate of $1e{-}6$ for Resnet101 visual encoder, $5e{-}6$ for RoBERTa text encoder and $1e{-}5$ for rest of the layers. We pre-train on 64 V100s with a batch size of 1, and finetune on RefCOCO/RefCOCO+ with a batch size of 2 on 16 V100s. The total training time is included in the respective tables. We will release pre-trained checkpoints for both 3D and 2D models.

A.2 Negative training with detection phrases

We devise object detection as language grounding of an utterance formed by concatenating a sequence of category labels, e.g. “Chair. Dining table. Bed. Plant. Sofa.”. The task is again to i) detect the mentioned objects in the scene, i.e. return bounding boxes of their instances, and ii) associate each localized box to a span, i.e. an object category in the utterance.

To form these detection phrases, one solution could be to concatenate all object classes into a long utterance. However, this can be impractical if the domain-vocabulary is “open”, or, in practice, very large (485 classes in ScanNet, 1600 in Visual Genome and so on). Instead, assuming that we have object annotations, we sample out of the positive labels that are annotated for a scene and a number of negative ones, corresponding to class names that do not appear in the scene. Having negative classes in the detection phrases helps the precision of the model, as it learns not to fire for every noun phrase that appears in an utterance. More specifically, the text-query contrastive losses described in the method section push the negative class’ text representation away from the query representation of existing objects.

MDETR also considers an object detection evaluation. However, there are two noticeable differences. First, they use only single-category utterances, e.g. “Dog.”. This category can be either positive (appears in the annotations) or negative (does not appear in the annotations), according to a sampling ratio. Opposite to that, our detection phrases are longer, consisting of multiple object categories, both positive and negative. Second, MDETR employs these sentences after pre-training, to train and evaluate their model as an object detector. Instead, we mix detection phrases through the training, leading to considerable quantitative gains in both 3D and 2D.

Lastly, although the ratio r of positive to negative classes that appear in a detection phrase is a hyperparameter, we report results only for $r = 2$ and sample at most 8 positive classes. We leave tuning of this hyperparameter for future research.

A.3 Additional ablations on 3D

We first ablate on how to encode the bounding box stream. We consider three options: i) bounding box features alone, ii) bounding box features and soft logits obtained

Model	Accuracy
box features only	44.2
box features and logits	43.1
box features and class embeddings	**48.5**

Table A.1: **Ablation for BUTD-DETR on the SR3D validation set**. We compare between different box encoding choices.

Model	Accuracy
separate stream	45.7
concatenated to point cloud	**48.5**

Table A.2: **Ablation for BUTD-DETR on the SR3D validation set**. We compare two fusion techniques between the visual and box stream in the encoder.

from the detector, iii) bounding box features and class embeddings, which is the approach we use in BUTD-DETR . In all cases, the layers used to encode the boxes are the same, as described in the results section. To encode logits, we apply softmax and a linear layer that map the 485-d vectors to 32-d. Then we use this vector as the "class embedding", identical to how we handle class embeddings in case iii. The comparison is shown in Table A.1. Combining box features and class embeddings gives the best performance. The model that uses logits underperforms, possibly because the predicted logits for training and testing come from different distributions: the detector is overconfident in the training set (giving more peaky distributions) but less confident on the test set (resulting in more smoothed distributions).

We also ablate on how to attend to the box stream in the encoder phase. We experiment with a) having boxes as a separate stream and allowing visual tokens to cross-attend to it; or b) append directly to every point in the cloud the features of a box that contain it, padding with zeros for points that do not lie inside any box, as described in A.1, which is what we use in the 3D version of BUTD-DETR . Appending features to the point cloud works better, but attending to a separate stream of boxes still largely outperforms the highest-performing competitor in the literature (LanguageRefer with 39.5%). In our 2D implementation, appending box features to pixels did not work, probably because of significant overlap between multiple object proposals.

A.4 Additional Results on 2D Language Grounding

We test BUTD-DETR on Flickr30k entities dataset [265]. Given an utterance about an image, the task is to predict bounding boxes for all the objects mentioned. Note that unlike the referential grounding task we show in our main results that is evaluated on RefCOCO/RefCOCO+, the task here is to find *all* objects mentioned in the utterance and not only the root object. Following MDETR, we directly evaluate our pre-trained model on Flickr30k without any further fine-tuning. For evaluation, we follow the ANY-BOX protocol [181] and evaluate our performance in terms of Recall metric on standard val and test splits. The results are shown in Table-2.4. We achieve results comparable with state-of-the-art MDETR model while converging in less than half the number of GPU hours.

A.5 More qualitative results

We show qualitative results of the 2D version of BUTD-DETR on RefCOCO in Figure A.1. We also show failure cases on SR3D in Figure A.2. More qualitative results on NR3D shown in A.3.

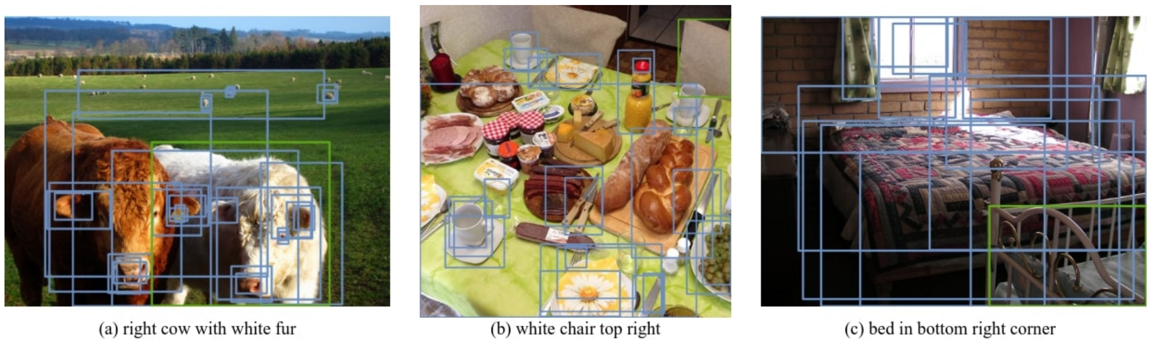
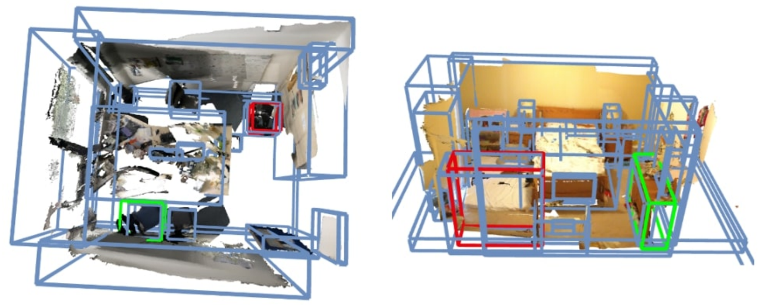

(a) right cow with white fur
(b) white chair top right
(c) bed in bottom right corner

Figure A.1: **Qualitative results of BUTD-DETR on RefCOCO.** The detector's proposals are shown in blue, our model's prediction in green. BUTD-DETR can predict boxes that the detector misses, e.g. in (b), the chair is missed by the detector so none of the previous detection-bottlenecked approaches could ground this phrase. In (a) and (c) the detector succeeds with low IoU but BUTD-DETR is able to predict a tight box around the referent object.

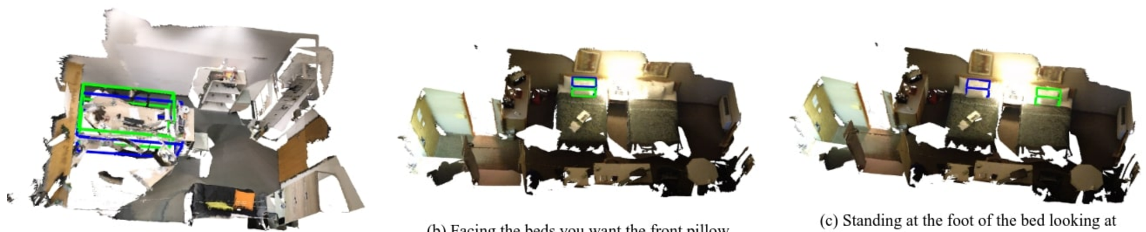


(a) the office chair that is beside the chair

(b) find the dresser that is next to the trash can

(c) find the armchair that is next to the table

Figure A.2: Failure cases of BEAUTY-DETR on SR3D. Our predictions with red, ground-truth with green. Even if the box is there, still our model can fail, proving that ranking the correct boxes over other proposals remains a hard problem.



(a) choose the desk on the opposite side of the couch

(b) Facing the beds you want the front pillow on the left bed.

(c) Standing at the foot of the bed looking at the head on the bed. It is the pillow in the from on the bed on the right.

Figure A.3: Qualitative results of BUTD-DETR on NR3D. Our predictions are shown blue, ground-truth in green. The language of NR3D is more complex and the utterances are longer. Case (c) is a failure case.

Appendix B

ODIN: A Single Model for 2D and 3D Segmentation

B.1 Experiments

B.1.1 Evaluations on ScanNet and ScanNet200 Hidden Test Sets

We submit ODIN to official test benchmarks of ScanNet [47] and ScanNet200 [294]. Following prior works, we train ODIN on a combination of train and validation scenes. Unlike some approaches that employ additional tricks like DBSCAN [303], ensembling models [178], additional specialized augmentations [360], additional pre-training on other datasets [377], finer grid sizes [348] and multiple forward passes through points belonging to the same voxel, our method avoid any such bells and whistles.

The results are shown in tables B.1,B.2,B.3,B.4. All conclusions from results on the validation set of these datasets as discussed in the main chapter are applicable here. On the ScanNet benchmark, ODIN achieves close to SOTA performance on semantic segmentation and mAP25 metric of Instance Segmentation while being significantly worse on mAP metric due to misalignments between sensor and mesh sampled point clouds. On ScanNet200 benchmark, ODIN sets a new SOTA on semantic segmentation and mAP50/mAP25 metric of Instance Segmentation, while

Point Cloud	Model	mAP	mAP50	mAP25
Sensor RGBD	ODIN-Swin-B (Ours)	**47.7**	**71.2**	**86.2**
Mesh Sampled	SoftGroup [344]	50.4	76.1	86.5
Mesh Sampled	PBNet [406]	57.3	74.7	82.5
Mesh Sampled	Mask3D [303]	56.6	78.0	87.0
Mesh Sampled	QueryFormer [221]	58.3	**78.7**	**87.4**
Mesh Sampled	MAFT [179]	**59.6**	78.6	86.0

Table B.1: **Comparison on ScanNet for Instance Segmentation Task.**

Point Cloud	Model	mIoU
Sensor RGBD	MVPNet [144]	64.1
Sensor RGBD	BPNet [120]	**74.9**
Sensor RGBD	DeepViewAgg [289]	-
Sensor RGBD	ODIN-Swin-B (Ours)	74.4
Rendered RGBD	VMVF [177]	**74.6**
Mesh Sampled	Point Transformer v2 [360]	75.2
Mesh Sampled	Stratified Transformer [178]	74.7
Mesh Sampled	OctFormer [348]	76.6
Mesh Sampled	Swin3D-L [377]	**77.9**
Mesh Sampled (Zero-Shot)	OpenScene [262]	-

Table B.2: **Comparison on ScanNet for Semantic Segmentation Task.**

achieving close to SOTA performance on mAP metric. Notably ODIN is the first method that operates over sensor RGB-D data for instance segmentation and achieves competitive performance to models operating over mesh-sampled point clouds.

B.1.2 Evaluation on S3DIS and Matterport3D

We also benchmark ODIN on Matterport3D [24] and S3DIS [8] datasets.

Matterport: Matterport3D comprises 90 building-scale scenes, further divided into individual rooms, with 1554 training rooms and 234 validation rooms. The dataset provides a mapping from each room to the camera IDs that captured images for that room. After discarding 158 training rooms and 18 validation rooms without a valid camera mapping, we are left with 1396 training rooms and 158 validation rooms. For instance segmentation results, we train the state-of-the-art Mask3D [303] model on the same data (reduced set after discarding invalid rooms). For semantic segmentation, we conduct training and testing on the reduced set, while baseline

Point Cloud	Model	mAP	mAP50	mAP25
Sensor RGBD	ODIN-Swin-B (Ours)	**27.2**	**39.4**	**47.5**
Mesh Sampled	Mask3D [303]	**27.8**	**38.8**	**44.5**
Mesh Sampled	QueryFormer [221]	-	-	-
Mesh Sampled	MAFT [179]	-	-	-
Mesh Sampled (Zero-Shot)	OpenMask3D [331]	-	-	-

Table B.3: **Comparison on ScanNet200 for Instance Segmentation Task.**

Point Cloud	Model	mIoU
Sensor RGBD	ODIN-Swin-B (Ours)	**36.8**
Mesh Sampled	LGround [294]	27.2
Mesh Sampled	CeCo [408]	**34.0**
Mesh Sampled	Octformer [348]	32.6

Table B.4: **Comparison on ScanNet200 for Semantic Segmentation Task.**

numbers are taken from the OpenScene [262] paper, trained and tested on the original data. Given the small size of the discarded data, we do not anticipate significant performance differences. The official benchmark of Matterport3D tests on 21 classes; however, OpenScene also evaluates on 160 classes to compare with state-of-the-art models on long-tail distributions. We follow them and report results in both settings.

S3DIS: S3DIS comprises 6 building-scale scenes, typically divided into 5 for training and 1 for testing. The dataset provides raw RGB-D images, captured panorama images, and images rendered from the mesh obtained after reconstructing the original sensor data. Unlike Matterport3D, S3DIS do not provide undistorted raw images; thus, we use the provided rendered RGB-D images. Some rooms in S3DIS have major misalignments between RGB-D images and point clouds, which we partially address by incorporating fixes from DeepViewAgg [289] and introducing our own adjustments. Despite these fixes, certain scenes still exhibit significantly low overlap between RGB-D images and the provided mesh-sampled point cloud. To mitigate this, we query images from other rooms and verify their overlap with the provided point cloud for a room. This partially helps in addressing the low overlap issue.

The official S3DIS benchmark evaluates 13 classes. Due to the dataset's small size,

		21		160	
Point Cloud	Model	mAP	mAP25	mAP	mAP25
Sensor RGBD	Mask3D [303]	7.2	16.8	2.5	10.9
Sensor RGBD	ODIN-ResNet50 (Ours)	22.5	56.4	11.5	27.6
Sensor RGBD	ODIN-Swin-B (Ours)	**24.7**	**63.8**	**14.5**	**36.8**
Mesh Sampled	Mask3D [303]	**22.9**	**55.9**	**11.3**	**23.9**

Table B.5: **Comparison on Matterport3D for Instance Segmentation Task.**

		21		160	
Point Cloud	Model	mIoU	mAcc	mIoU	mAcc
Sensor RGBD	ODIN-ResNet50 (Ours)	54.5	65.8	22.4	28.5
Sensor RGBD	ODIN-Swin-B (Ours)	**57.3**	**69.4**	**28.6**	**38.2**
Mesh Sampled	TextureNet [123]	-	63.0	-	-
Mesh Sampled	DCM-Net [302]	-	**67.2**	-	-
Mesh Sampled	MinkowskiNet [41]	**54.2**	64.6	-	**18.4**
Mesh Sampled (Zero-Shot)	OpenScene [262]	**42.6**	**59.2**	-	**23.1**

Table B.6: **Comparison on Matterport3D for Semantic Segmentation Task.**

some models pre-train on additional datasets like ScanNet, as seen in SoftGroup [344], and on Structured3D datasets [407], consisting of 21,835 rooms, as done by Swin3D-L [377]. Similar to Mask3D [303], we report results in both settings of training from scratch and starting from weights trained on ScanNet.

Like ScanNet and ScanNet200, both S3DIS and Matterport3D undergo post-processing of collected RGB-D data to construct a mesh, from which a point cloud is sampled and labeled. Hence, we train both Mask3D [303] and our model using RGB-D sensor point cloud data and evaluate on the benchmark-provided point cloud. Additionally, we explore model variants by training and testing them on the mesh-sampled point cloud for comparative analysis.

We draw the following conclusions:

ODIN outperforms SOTA 3D models on Matterport3D Instance Segmentation Benchmark across all settings (table B.5)

ODIN sets a new state-of-the-art on Matterport3D Semantic Segmentation Benchmark (table B.6): Our model achieves superior performance in both the 21 and 160 class settings. It also largely outperforms OpenScene [262] on both settings. OpenScene is a zero-shot method while ODIN is supervised in-domain, making this comparison unfair. However, OpenScene notes that their zero-shot model outperforms fully-supervised

Point Cloud	Model	mAP	mAP50	mAP25
Sensor RGBD	Mask3D [303]	40.7	54.6	64.2
	Mask3D [303] †	41.3	55.9	66.1
Sensor RGBD	ODIN-ResNet50 (Ours)	36.3	48.0	61.2
Sensor RGBD	ODIN-ResNet50 † (Ours)	**44.7**	**57.7**	67.5
Sensor RGBD	ODIN-Swin-B † (Ours)	43.0	56.4	**70.0**
Mesh Sampled	SoftGroup [344] †	51.6	66.1	-
Mesh Sampled	Mask3D [303]	56.6	68.4	75.2
Mesh Sampled	Mask3D [303] †	**57.8**	**71.9**	**77.2**
Mesh Sampled	QueryFormer [221]	57.7	69.9	-
Mesh Sampled	MAFT [179]	-	69.1	75.7

Table B.7: **Comparison on S3DIS Area5 for Instance Segmentation Task.** († = uses additional data)

Point Cloud	Model	mIoU
Sensor RGBD	MVPNet [144]	62.4
Sensor RGBD	VMVF [177]	65.4
Sensor RGBD	DeepViewAgg [289]	67.2
Sensor RGBD	ODIN-ResNet50 (Ours)	59.7
Sensor RGBD	ODIN-ResNet50 † (Ours)	66.8
Sensor RGBD	ODIN-Swin-B † (Ours)	**68.6**
Mesh Sampled	Point Transformer v2 [360]	71.6
Mesh Sampled	Stratified Transformer [178]	72.0
Mesh Sampled	Swin3D-L [377] †	**74.5**

Table B.8: **Comparison on S3DIS for Semantic Segmentation Task.** († = uses additional data)

models in 160 class setup as their model is robust to rare classes while the supervised models can severely suffer in segmenting long-tail. ConceptFusion [146], another open-vocabulary 3D segmentation model, also draws a similar conclusion. With this result, we point to a possibility of supervising in 3D while also being robust to long-tail by simply utilizing the strong 2D pre-trained weight initialization.

On S3DIS Instance Segmentation Benchmark (table B.7), in the setup where baseline Mask3D start from ScanNet pre-trained checkpoint, our model outperforms them in the RGBD point cloud setup but obtains lower performance compared to mesh sampled point cloud methods and when compared on the setup where all models train from scratch.

On S3DIS Semantic Segmentation Benchmark (table B.8, ODIN trained with

Input	Model	All			Head			Common			Tail		
		mAP	mAP50	mAP25	mAP	mAP50	mAP25	mAP	mAP50	mAP25	mAP	mAP50	mAP25
Sensor RGBD point cloud	Mask3D § [303]	15.5	21.4	24.3	21.9	31.4	37.1	13.0	17.2	18.9	7.9	10.3	11.5
	ODIN-ResNet50 (Ours)	25.6	36.9	43.8	34.8	51.1	63.9	23.4	33.4	37.9	17.8	24.9	28.1
	ODIN-Swin-B (Ours)	**31.5**	**45.3**	**53.1**	37.5	54.2	**66.1**	**31.6**	**43.9**	**50.2**	**24.1**	**36.6**	**41.2**
Mesh Sampled point cloud	Mask3D [303]	27.4	37.0	42.3	**40.3**	**55.0**	62.2	22.4	30.6	35.4	18.2	23.2	27.0

Table B.9: **Detailed ScanNet200 results for Instance Segmentation** (§ = trained by us using official codebase)

ScanNet weight initialization outperforms all RGBD point cloud based methods, while achieving competitive performance on mesh sampled point cloud. When trained from scratch, it is much worse than other baselines. Given the limited dataset size of S3DIS with only 200 training scenes, we observe severe overfitting.

B.1.3 ScanNet200 Detailed Results

ScanNet200 [294] categorizes its 200 object classes into three groups—*Head*, *Common*, and *Tail*—each comprising 66, 68, and 66 categories, respectively. In table B.9, we provide a detailed breakdown of the ScanNet200 results across these splits. We observe that in comparison to SOTA Mask3D model trained on mesh-sampled point cloud, ODIN achieves lower performance on *Head* classes, while significantly better performance on *Common* and *Tail* classes. This highlights the contribution of effectively utilizing 2D pre-trained features, particularly in detecting a long tail of class distribution where limited 3D data is available.

B.1.4 Variation of Performance with Number of Views

We examine the influence of the number of views on segmentation performance using the AI2THOR dataset, specifically focusing on the 2D mAP performance metric. The evaluation is conducted by varying the number of *context* images surrounding a given *query* RGB image. Starting from a single-view without any context (N=0), we increment N to 5, 10, 20, 40, 60, and finally consider all images in the scene as context. ODIN takes these $N + 1$ RGB-D images as input, predicts per-pixel instance segmentation for each image, and assesses the 2D mAP performance on the *query* image. The results, depicted in fig. B.1, show a continuous increase in 2D mAP with the growing number of views. This observation underscores the advantage of utilizing

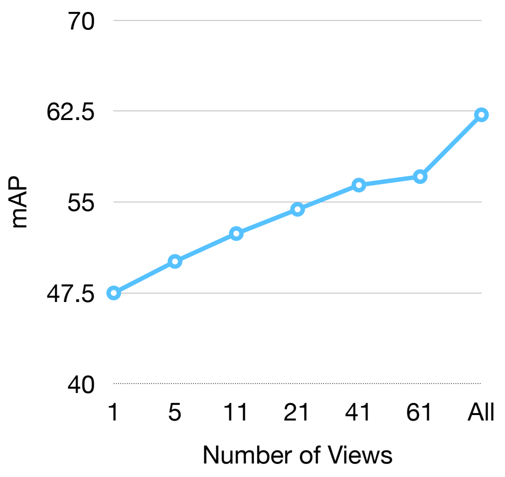


Figure B.1: **2D mAP Performance Variation with increasing number of context views used**

multiview RGB-D images over single-view RGB images whenever feasible.

B.1.5 Inference Time

We assess the inference time of Mask3D and ODIN by averaging the forward pass time of each model across the entire validation set, utilizing a 40 GB VRAM A100. When fed the mesh-sampled point cloud directly, Mask3D achieves an inference time of 228ms. When provided with the sensor point cloud as input, the inference time increases to 864 ms. Mask3D with sensor point cloud is slower than with mesh point cloud because at the same voxel size (0.02m), more voxels are occupied in sensor point cloud (110k on avg.) compared to mesh point clouds (64k on avg.) as mesh-cleaning sometimes discards large portion of the scene. The transfer of features from the sensor point cloud to the mesh point cloud adds an extra 7 ms. ODIN-SwinB, which operates over the sensor point cloud, has an inference time of 960ms.

B.2 Additional Implementation Details

The detailed components of our architecture and their descriptions are presented in fig. B.2.

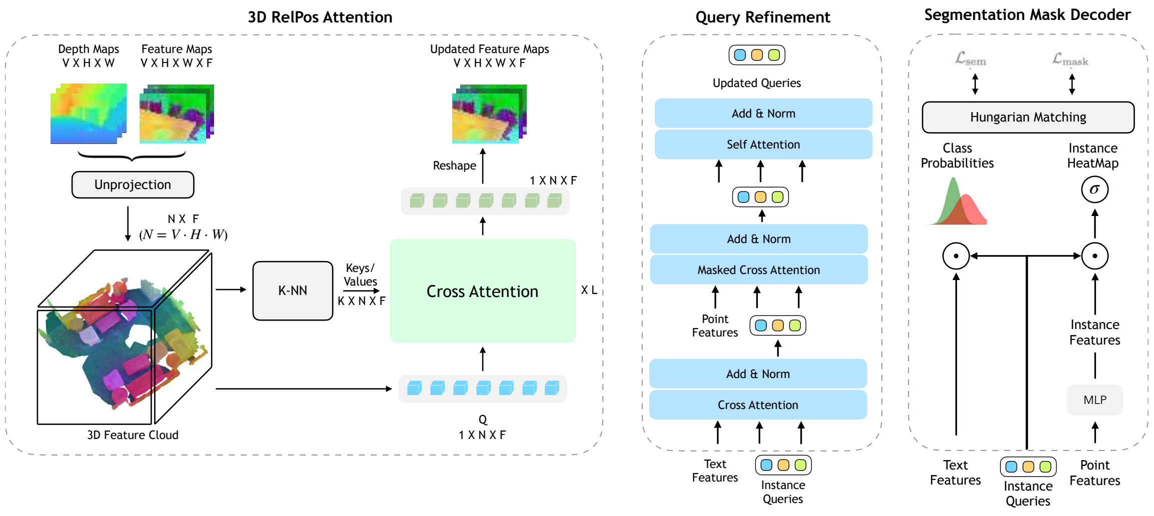


Figure B.2: **Detailed ODIN Architecture Components**: On the **Left** is the 3D RelPos Attention module which takes as input the depth, camera parameters and feature maps from all views, lifts the features to 3D to get 3D tokens. Each 3D token serves as a query. The K-Nearest Neighbors of each 3D token become the corresponding keys and values. The 3D tokens attend to their neighbours for L layers and update themselves. Finally, the 3D tokens are mapped back to the 2D feature map by simply reshaping the 3D feature cloud to 2D multi-view feature maps. On the **Middle** is the query refinement block where queries first attend to the text tokens, then to the visual tokens and finally undergo self-attention. The text features are optional and are only used in the open-vocabulary decoder setup. On the **Right** is the segmentation mask decoder head where the queries simply perform a dot-product with visual tokens to decode the segmentation heatmap, which can be thresholded to obtain the segmentation mask. In the Open-Vocabulary decoding setup, the queries also perform a dot-product with text tokens to decode a distribution over individual words. In a closed vocabulary decoding setup, queries simply pass through an MLP to predict a distribution over classes.

More implementation details are presented below:

Augmentations: For RGB image augmentation, we implement the Large Scale Jittering Augmentation method from Mask2Former [36], resizing images to a scale between 0.1 and 2.0. We adjust intrinsics accordingly post-augmentation and apply color jittering to RGB images. Training involves a consecutive set of N images, typically set to 25. With a 50% probability, we randomly sample k images from the range $[1, N]$ instead of using all N images. Additionally, instead of consistently sampling N consecutive images, we randomly skip k images in between, where k ranges from 1 to 4.

For 3D augmentations, we adopt the Mask3D [303] approach, applying random 3D rotation, scaling, and jitter noise to the unprojected XYZs. Elastic distortion and random flipping augmentations from Mask3D are omitted due to a slight drop in performance observed in our initial experiments.

Image Resolutions We use a resolution of 256×256 for ScanNet, 512×512 for ScanNet200, and AI2THOR. In our AI2THOR experiments, we discovered that employing higher image resolutions enhances the detection of smaller objects, with no noticeable impact on the detection of larger ScanNet-like objects. This observation was confirmed in ScanNet, where we experimented with 512×512 image resolutions and did not observe any discernible benefit.

Interpolation Throughout our model, interpolations are employed in various instances, such as when upsampling the feature map from 1/8th resolution to 1/4th. In cases involving depth, we unproject feature maps to 3D and perform trilinear interpolation, as opposed to directly applying bilinear interpolation on the 2D feature maps. For upsampling/downsampling the depth maps, we use the nearest interpolation. Trilinear interpolation proves crucial for obtaining accurate feature maps, particularly at 2D object boundaries like table and floor edges. This is because nearest depth interpolation may capture depth from either the table or the floor. Utilizing trilinear upsampling of feature maps ensures that if the upsampled depth is derived from the floor, it interpolates features from floor points rather than table points.

Use of Segments: Some datasets, such as ScanNet and ScanNet200, provide supervoxelization of the point cloud, commonly referred to as *segments*. Rather than directly segmenting all input points, many 3D methods predict outputs over these segments. Specifically, Mask3D [303] featurizes the input points and then conducts mean pooling over the features of points belonging to a segment, resulting in one feature per segment. Following prior work, we also leverage segments in a similar manner. We observe that utilizing segments is crucial for achieving good mAP performance, while it has no discernible impact on mAP25 performance. We suspect that this phenomenon may arise from the annotation process of these datasets. Humans were tasked with labelling segments rather than individual points, ensuring that all points within a segment share the same label. Utilizing segments with our models guarantees that the entire segment is labelled with the same class. It's worth noting that in AI2THOR, our method and the baselines do not utilize these segments, as they are not available.

Post-hoc output transfer vs feature transfer: ODIN takes the sensor point cloud as input and generates segmentation output on the benchmark-provided point cloud. In this process, we featurize the sensor point cloud and transfer these features from the sensor point cloud to the benchmark-provided point cloud. Subsequently, we predict segmentation outputs on this benchmark-provided feature cloud and supervise the model with the labels provided in the dataset. An alternative approach involves segmenting and supervising the sensor RGB-D point cloud and later transferring the segmentation output to the benchmark point cloud for evaluation. We experimented with both strategies and found them to yield similar results. However, as many datasets provide segmentation outputs only on the point cloud, transferring labels to RGB-D images for the latter strategy requires careful consideration. This is due to the sparser nature of the provided point cloud compared to the RGB-D sensor point cloud, and factors such as depth noise and misalignments can contribute to low-quality label transfer. Consequently, we opt for the former strategy in all our experiments.

Depth Hole-Infilling: The sensor-collected depth maps usually have holes around object boundaries and shiny/transparent surfaces. We perform simple OpenCV depth inpainting to fill these holes. We tried using neural-based depth completion methods and NERF depth-inpainting but did not observe significant benefits.

AI2THOR Data Collection: AI2THOR [174] is an embodied simulator where an agent can navigate within a house, execute actions, and capture RGB-D images of the scene. We load the structurally generated houses from ProcTHOR [54] into the AI2THOR simulator, and place an agent randomly at a navigable point provided by the simulator. The agent performs a single random rotation around its initial location and captures an RGB-D frame. This process is repeated, with the agent spawning at another random location, until either all navigable points are exhausted or a maximum of $N = 120$ frames is collected. While ProcTHOR offers 10,000 scenes, we randomly select only 1,500 scenes to match the size of ScanNet. Additionally, we retain scenes with fewer than 100 objects, as our model utilizes a maximum of 100 object queries.

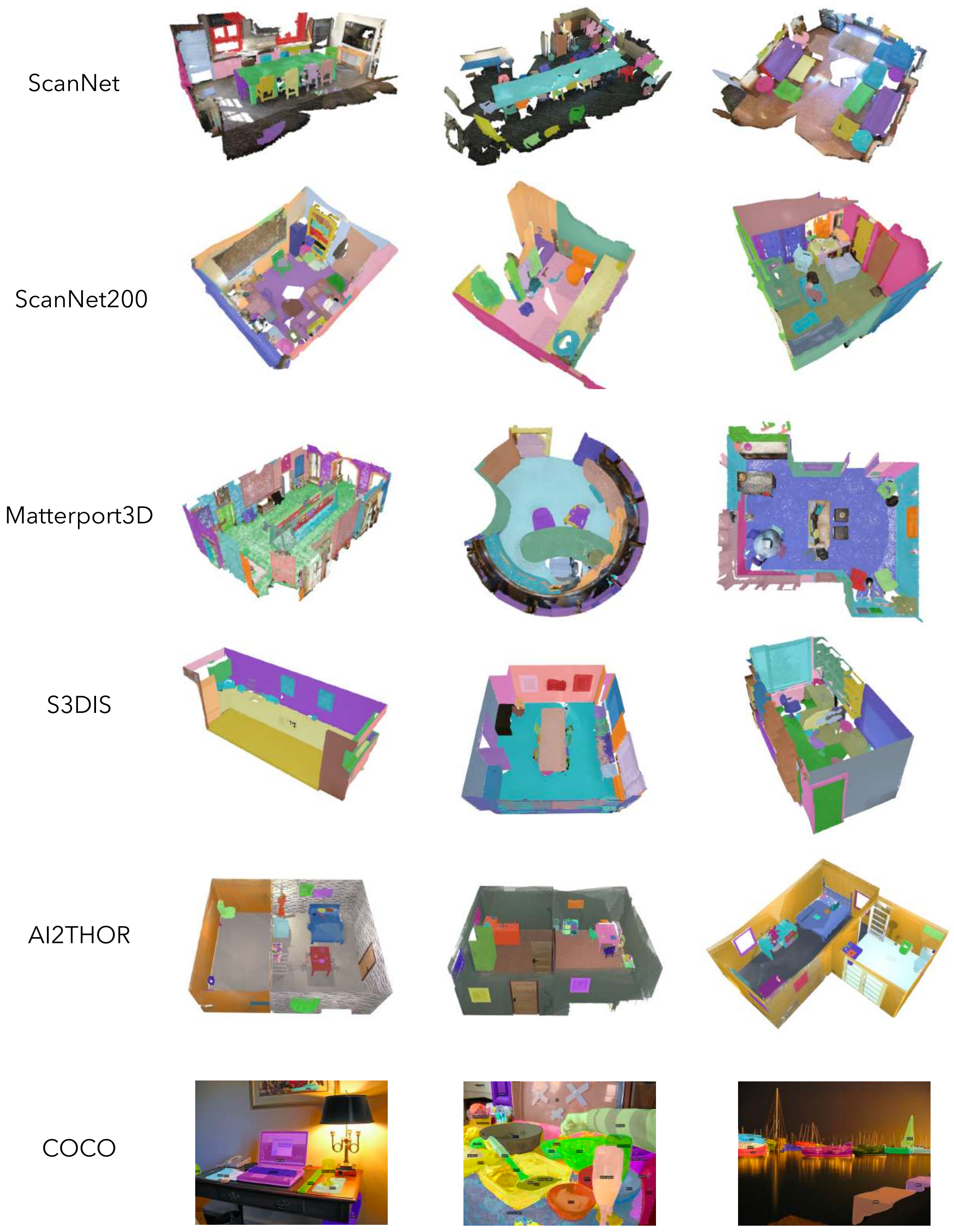


Figure B.3: Qualitative Results on various 3D and 2D datasets

B.3 Qualitative Results

fig. B.3 shows qualitative visualizations of ODIN for various 3D and 2D datasets.

Appendix C

Analogy-Forming Transformers for Few-Shot 3D Parsing

Our Appendix is organized as follows: In Section C.1, we provide implementation details and pseudo-code for training Analogical Networks. In Section C.2 we ablate Analogical Networks' performance under varying memory retrieval schemes in 5-shot setting and show qualitative results for the retriever in Section C.3. We show more results on noisy point clouds (ScanObjectNN [341] dataset) in C.4. We provide extensive qualitative visual object parsing results for single and multi-memory variants of Analogical Networks in section C.5. Lastly, we discuss additional related literature in Section C.6.

C.1 Implementation Details and Training Pseudo Code

The modulator encodes the input scene S and each retrieved memory scene M into a set of 3D point features using PointNet++ backbone. We encode positional information using rotary 3D positional encodings [186, 325], which have the property that $P(x)^T P(y) = P(y - x)$, where P the positional encoding function and x, y two 3D points. These embeddings are multiplied with queries and keys in the attention operations, thus making attention translation-invariant. For memory queries, we use the centroid of the corresponding memory part to compute positional encodings. For scene-agnostic queries, we use the center of the input object.

Each labelled part p in each memory M is encoded into an 1D feature vector f_p^M by average pooling its point features. Next, the queries (memory and scene-agnostic combined) self-attend and cross-attend to the input point features and update themselves; point features self-attend and cross-attend to queries to also update themselves. This input feature update is a difference from existing detection transformers, where only the queries are updated. We consider 6 layers of self and cross-attention.

Lastly, we upsample the point features to the original resolution using convolutional layers [270] and compute an inner product between each query and point feature to compute the segmentation mask for each query. The output of the modulator is a set of N_q segmentation mask proposals and corresponding confidence scores, where N_q is the total number of queries. At training time, these proposals are matched to ground-truth instance binary masks using the Hungarian matching algorithm [23]. For the proposals that are matched to a ground-truth instance, we compute the segmentation loss, which is a per-point binary cross-entropy loss [35, 345]. We also supervise the confidence score of each query, similar to [23]. The target labels are 1 for the proposals matched with a ground-truth part and 0 for non-matched. We found it beneficial to apply these losses after every cross-attention layer in the modulator. At test time, we multiply the per point mask occupancy probability with tiled confidence scores to get a $N_p \times N_q$ tensor (N_p is the number of points); then each point is assigned to the highest scoring query by computing inner product and taking per-point argmax over the queries. The modulator's weights are trained with within-scene and cross-scene training where the modulating memories are sampled from the top-k retrieved memories. During cross-scene training, we further co-train with within-scene training data.

For both stages of training (i.e. within-scene correspondence pre-training and cross-scene training), we use AdamW optimizer [218] with an initial learning rate of 2e$-$4 and batch size of 16. We train the model for 100 epochs within-scene and 60 cross-scene. For few-shot fine-tuning/evaluation, we use AdamW optimizer with an initial learning rate of 3e$-$5 and batch size of 8. We fine-tune for 90 epochs and we report the performance across 10 different episodes, where each episode has

Algorithm 1 Pseudo code for within-scene correspondence pre-training of Analogical Networks

```
# S: input point cloud, M: memory point cloud, Np: numbers of points in S or M, N: sub-sampled points
    , C: number of feature channels, P: number of parts in M
# augment: a sequence of standard 3D point cloud augmentations
# pc_encoder: point cloud encoder
# Xp(M): ground-truth label assignment of points in parts, copied directly from the memory
# part_encoder: Computes the part features using mean pooling
# pos_encode: Adds positional encoding
# upsampler: Upsamples point cloud
# Segmentation_Loss: Cross entropy loss to assign each point to the Hungarian matched query.

for S in dataloader: # load a batch with B samples
   M = S # the memory is the un-augmented version
   S = augment(S) # the input is augmented
   # S : B x Np x 3 and M: B x Np x 3
   # Compute point features
   F^S = pc_encoder(S) # B x N x C
   F^M = pc_encoder(M) # B x N x C

   # Initialize memory part queries
   f^M = part_encoder(F^M) # B x P x C

   # Compute positional embedding
   x_pos = pos_encode(F^S)
   y_pos = pos_encode(f^M) # B x P x C

   Loss = 0
   # Do multiple layers of modulation using Self-Attn and Cross-Attn
   for layer in num_layers:
      x = Cross-Attn(x, y, x_pos, y_pos) # B x N x C
      y = Cross-Attn(y, x, y_pos, x_pos) # B x P x C
      x = Self-Attn(x, x_pos) # B x N x C
      y = Self-Attn(y, y_pos) # B x P x C

      X = upsampler(x) # B x Np x C
      point_query_similarity = matmul(normalize(X), normalize(y.T)) # B x Np x P

      Loss += Segmentation_Loss(argmax(point_query_similarity, -1), Xp(M))


   # optimizer step
   loss.backward()
   optimizer.step()
```

a different set of K support samples. We describe Analogical Networks' training details in pseudo-code for within (Algorithm 1) and cross-scene training (Algorithm 2) respectively.

For our `DETR3D` baseline we use same hyperparameters and train for 250 epochs. Training takes approximately 15 and 20 minutes per epoch on a single NVIDIA A100 gpu for `DETR3D` and Analogical Networks respectively. For the multi-memory model we reduce the batch size to 8 and the learning rate to 1e−4. Each epoch takes around 50 minutes.

Algorithm 2 Pseudo code for cross-scene training of Analogical Networks

```
# S: input point cloud, M: retrieved memory point cloud, Np: numbers of points in S or M, N: sub-
    sampled points, C: number of feature channels, P: number of parts in M, target_classes: semantic
     classes of ground-truth parts of S
# Q: number of learnable scene-agnostic queries
# pc_encoder: point cloud encoder
# Xp_Hungarian: Hungarian matched label assignment of points in S to queries
# yp_Hungarian: Hungarian matched label assignment of queries to GT parts in S
# matched: indices of queries that have been matched to a ground-truth part
# part_encoder: Computes the part features using mean pooling
# pos_encode: Adds positional encoding
# upsampler: Upsamples point cloud
# Objectness_Loss: Binary cross entropy loss to decide which queries (scene-agnostic+learnable) would
     be responsible for decoding parts
# Segmentation_Loss: Cross entropy loss to assign each point to the hungarian matched query.
# Semantic_Loss: Cross entropy loss to map hungarian matched queries to semantic classes

for S,M in dataloader: # load a batch with B samples
    # S : B x Np x 3 and M: B x Np x 3
    # Compute point features
    F^S = pc_encoder(S) # B x N x C
    F^M = pc_encoder(M) # B x N x C

    # Initialize memory part queries
    f^M = part_encoder(F^M) # B x P x C

    # Compute positional embedding
    x_pos = pos_encode(F^S)
    y = pos_encode(Concatenate(f^M, scene_agnostic_queries)) # B x (P + Q) x C

    Loss = 0
    # Do multiple layers of modulation using Self-Attn and Cross-Attn
    for layer in num_layers:
        x = Cross-Attn(x, y, x_pos, y_pos) # B x N x C
        y = Cross-Attn(y, x, y_pos, x_pos) # B x P x C
        x = Self-Attn(x) # B x N x C
        y = Self-Attn(y) # B x P x C

        X = upsampler(x) # B x Np x C
        point_query_similarity = matmul(normalize(X), normalize(y.T)) # B x Np x (P + Q)

        Loss += Segmentation_Loss(armax(point_query_similarity, -1), Xp_Hungarian) + Objectness_Loss(y
             , yp_Hungarian) + Semantic_Loss(y[matched], target_classes)


    # optimizer step
    loss.backward()
    optimizer.step()
```

Method	Fine-tuned?	Modulating Memory	Novel Category: 5-shot ARI (↑)
AnalogicalNets single-mem	✗	Random category-constr.	48.5 ± 0.76
		Retriever category-constr.	53.8 ± 1.03
		Oracle	61.9 ± 1.22
AnalogicalNets single-mem	✓	Random category-constr.	53.5 ± 1.28
		Retriever category-constr.	58.3 ± 1.36
		Oracle	62.3 ± 0.66

Table C.1: **Ablations on ARI segmentation performance under varying retrieval schemes for 5-shot on 4 novel categories**.

Base Categories												Novel Categories			
Chair	Display	Storage Furniture	Bottle	Clock	Door	Ear phone	Faucet	Knife	Lamp	Trash Can	Vase	Table	Bed	Dishwasher	Refrigerator
6323	928	2269	436	554	225	228	648	327	2207	321	1076	8218	194	181	187

Table C.2: **Number of samples per category in the PartNet dataset [243].** Note that each sample has annotations for three levels of segmentation granularity.

C.2 Performance under Varying Retrieval Schemes

In the few-shot setting, we evaluate the performance of Analogical Networks under varying memory retrieval schemes in Table C.1. We compare against a hypothetical *oracle* retriever that can fetch the most helpful (in terms of resulting ARI) memory for each input. All examined retrievers are category-constrained, i.e., they have access to the object category of the input and retrieve an object of the same category. Our conclusions are as follows: **(i) Analogical Networks with an oracle memory retriever perform better than Analogical Networks using memories retrieved by the retriever**. This suggests that better training of our retriever or exploring its co-training with the rest of our model could have significant impact in improving its performance. **(ii) Considering any of the 5-shot exemplars randomly does worse than using memories retrieved by our retriever.**

C.3 Qualitative Performance of the Retriever

We show qualitative results of the retriever on multiple classes, both seen (Figure C.1) during training and unseen (Figure C.2). We observe the following: **(i) The retriever considers fine-grained object similarities and not only class information.** To illustrate this, we include two examples for the "Chair" and "Earphone" classes in Figure C.1, as well as the "Bed" and "Refrigerator" classes in Figure C.2. Different instances of the same category retrieve very different memories, that share both structural and semantic similarities with the respective input point cloud. **(ii) The retriever generalizes to novel classes, not seen during training**, as shown in Figure C.2.

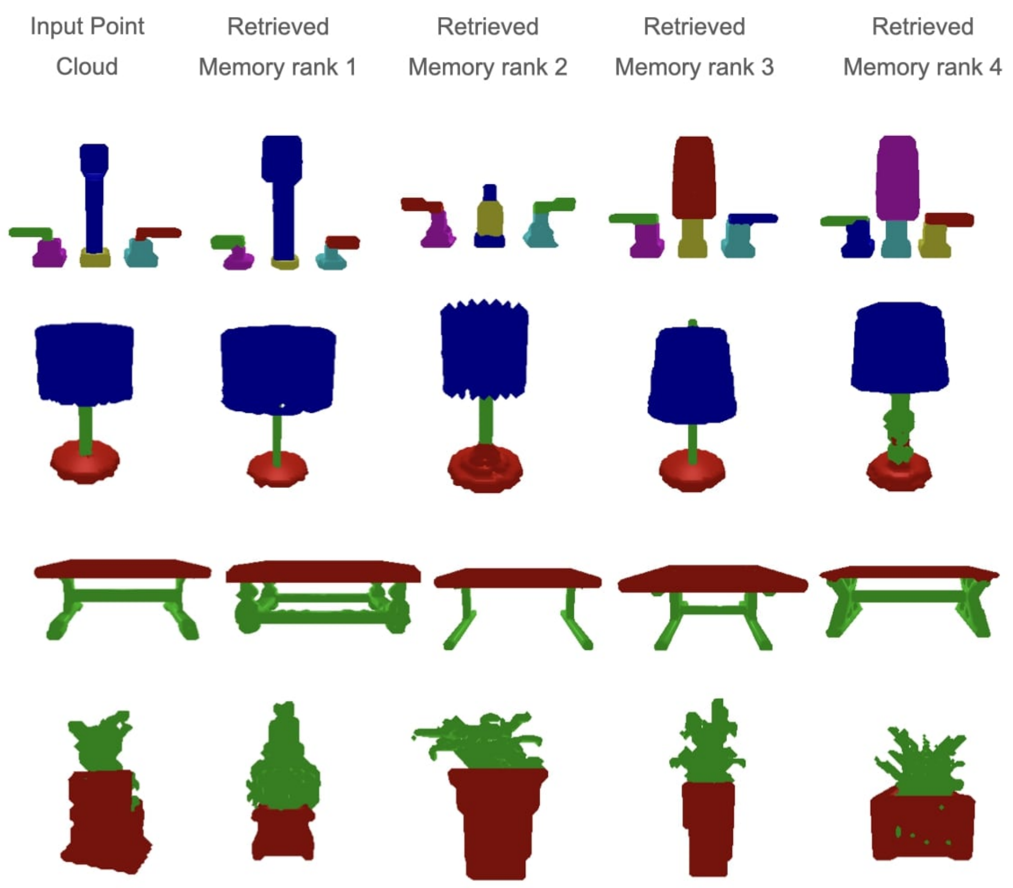


Figure C.1: Top-4 retrieved results for each input point cloud. Examples from base classes of PartNet [243] dataset. Note that instances of the same category can retrieve different memories, focusing on structural similarity and not only semantic.

C.4 Evaluation on ScanObjectNN Dataset

We test Analogical Networks on ScanObjectNN [341], which contains noisy and incomplete real-world point clouds. We split the training into 11 classes seen during training (bag, bin, box, cabinet, chair, desk, door, pillow, shelf, sink, sofa) and 4 unseen (bed, display, table, toilet). Note that ScanObjectNN is not consistently labelled. For example, as we show in rows 5 and 6 of Figure C.9, the legs of a chair may be annotated as a single part or multiple parts in the dataset. Although the PartNet dataset provides the "level" information, there is no such information in ScanObjectNN. Therefore we only qualitatively evaluate our model in Figure C.9.

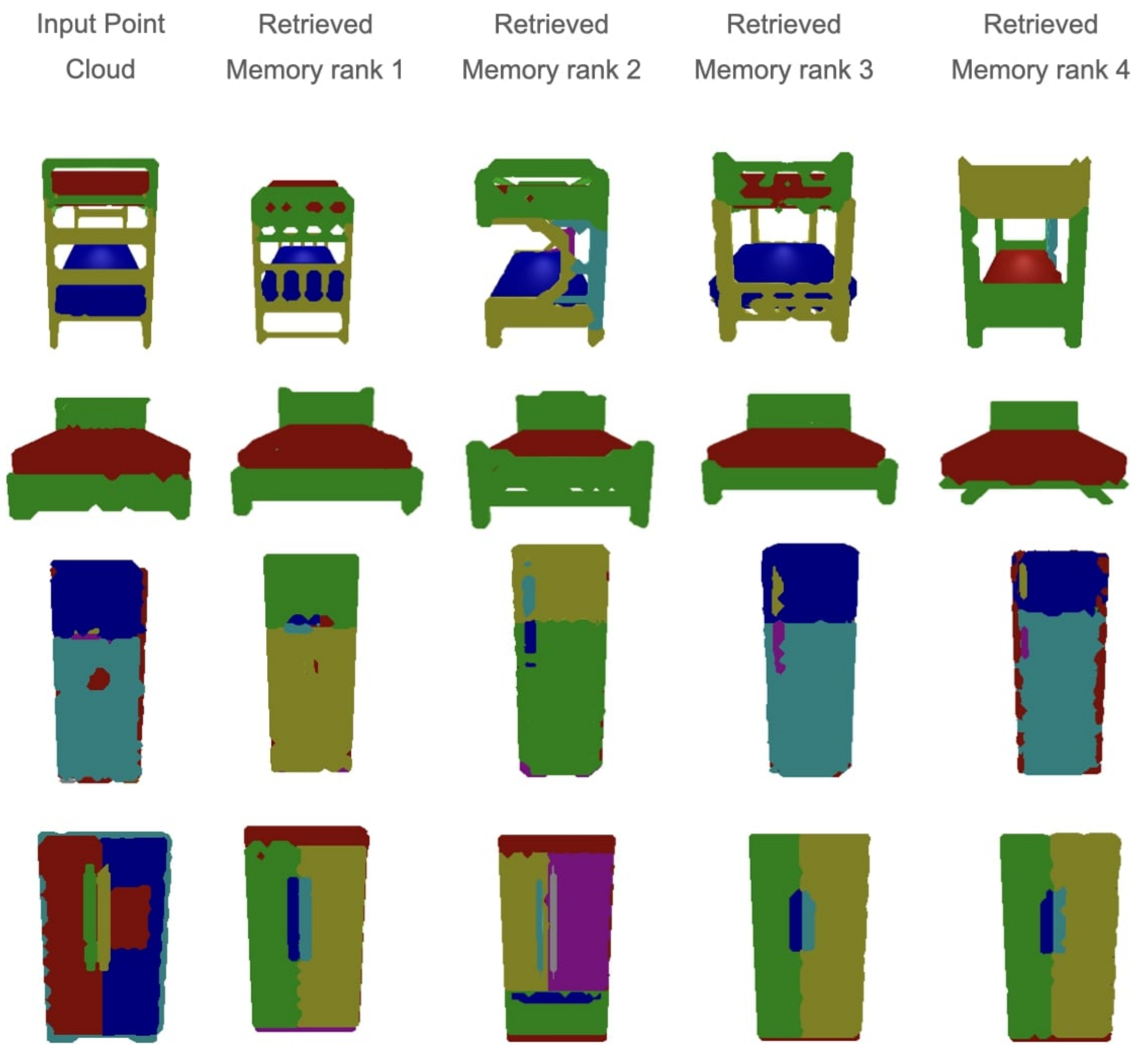


Figure C.2: Top-4 retrieved results for each input point cloud. Examples from novel classes of PartNet [243] dataset. Note that instances of the same category can retrieve different memories, focusing on structural similarity and not only semantic. This behavior generalizes to novel classes as well, even if the model has never seen such geometries before.

We can see that our predictions are always plausible and consistent with the retrieved memory, even if the expected label space is different.

C.5 Qualitative Results for Single-Memory and Multi-Memory Analogical Networks

In this section, we show our model's qualitative results for object parsing. In the Figures C.3, C.4, C.5, C.6, C.7, C.8, C.9 we use the following 5-column pattern:

- Unlabelled input point cloud
- Memory used for modulation
- Object parsing generated using only the memory part queries (not the scene-agnostic queries). In this column, regions that are not decoded by a memory part query are colored in black.
- Final predicted segmentation parsing using both memory-initialized queries and scene-agnostic queries. Regions that are colored in black in the third column but colored differently in the fourth column are decoded by scene-agnostic queries.
- Input point cloud's ground truth segmentation at the granularity level of the memory.

We qualitatively show the emergence of part correspondence between retrieved memory (column 2) and the input point cloud parsed using memory queries (column 3). Parts having the same color in columns 2 and 3 demonstrate correspondence, i.e. a part in column 2 decodes the part with the same color in column 3. Analogical Networks promote correspondence of parts on both base (Figure C.3 bottom and C.4) and novel (Figure C.3 top and C.5) categories. This correspondence is semantic but also geometric, as can be seen in Figure C.7. When multiple memories are available, Analogical Networks mix and match parts of different memories to parse the input. Furthermore, we show parsing results for `AnalogicalNets single-mem w/o within-scene` in Figure C.8. We observe that all of memory part query are inactive in the parsing stage. This demonstrates the utility of within-scene pre-training, as without this pre-training part correspondence does not emerge, as shown in Figure C.8. Lastly, we show that Analogical Networks generalize to noisy and incomplete point clouds in ScanObjectNN.

C.6 Additional related work

Neural-symbolic models Analogical Networks are a type of neural-symbolic model that represents knowledge explicitly, in terms of structured visual memories, where each one is a graph of part-entity neural embeddings. A structured visual memory can be considered the neural equivalent of a FRAME introduced in [240], *"a graph of nodes and their relations for representing a stereotyped situation, like being in a certain kind of living room, or going to a child's birthday party"*, to quote Minsky [240]. FRAME nodes would operate as "slots" to be filled with specific entities, or symbols, in the visual scene. Symbol detection would be carried out by a separate state estimation process such as general-purpose object detectors [410], employed also by recent neuro-symbolic models [233, 383, 384]. In-the-wild detection of symbols (e.g., chair handles, faucet tips, fridge doors) typically fails, which is the reason why these earlier symbolic systems of knowledge, largely disconnected from the sensory input, have not been widely adopted. Analogical Networks take a step towards addressing these shortcomings by including symbol detection as part of inference itself, through a top-down modulation that uses the context represented in the memory graph, to jointly search for multiple entities and localize them in context of one another.

3D instance segmentation has been traditionally approached as a clustering problem [31, 314]. Point-based methods learn either translation vectors mapping every point to its instance's center [28, 149, 344] or similarities across points [350, 397], followed by one or more stages of clustering. Similarly, [154, 351] oversegment the point cloud into small regions and then merge them into parts. Yu et al. [385] recursively decompose a point cloud into segments of finer resolution. [243, 328] learn representative vectors that form clusters by voting for each point. However, these approaches usually assume a fixed label space and need to train a separate model for each sub-task. In contrast, we employ Detection Transformers [23] for instance segmentation by repurposing the query vectors to act as representative vectors. We extend this set of queries with memory-initialized queries, enabling in-context reasoning. This allows us to train one model across all categories. As our results show, in absence of such memory contextualization, training one model across multiple categories hurts generalization (Table 4.3).

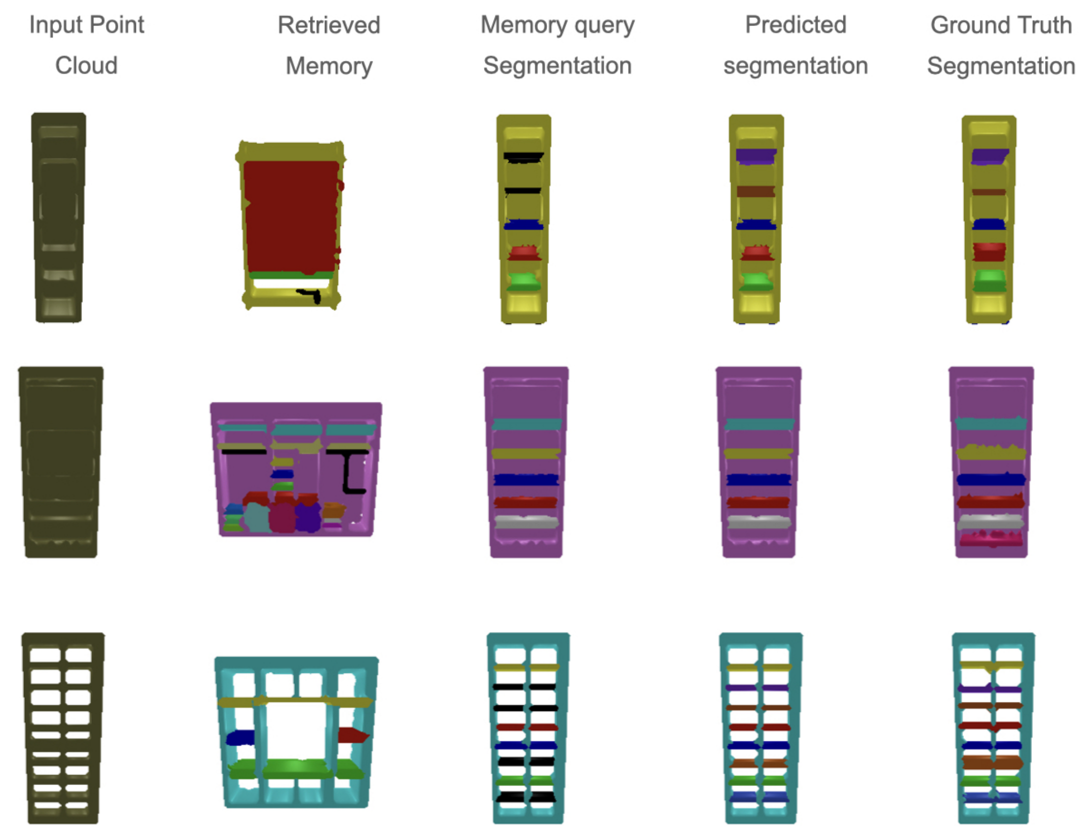


Figure C.3: More qualitative object parsing results using Analogical Networks.

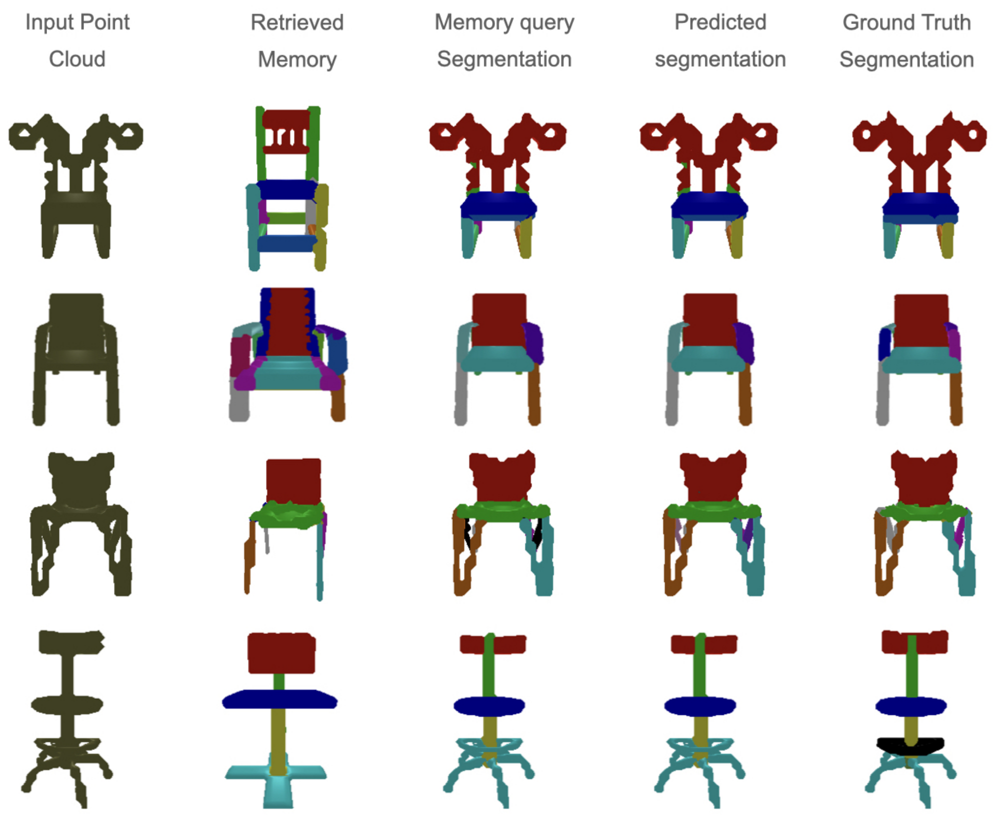


Figure C.4: More qualitative object parsing results that are predicted by Analogical Networks.

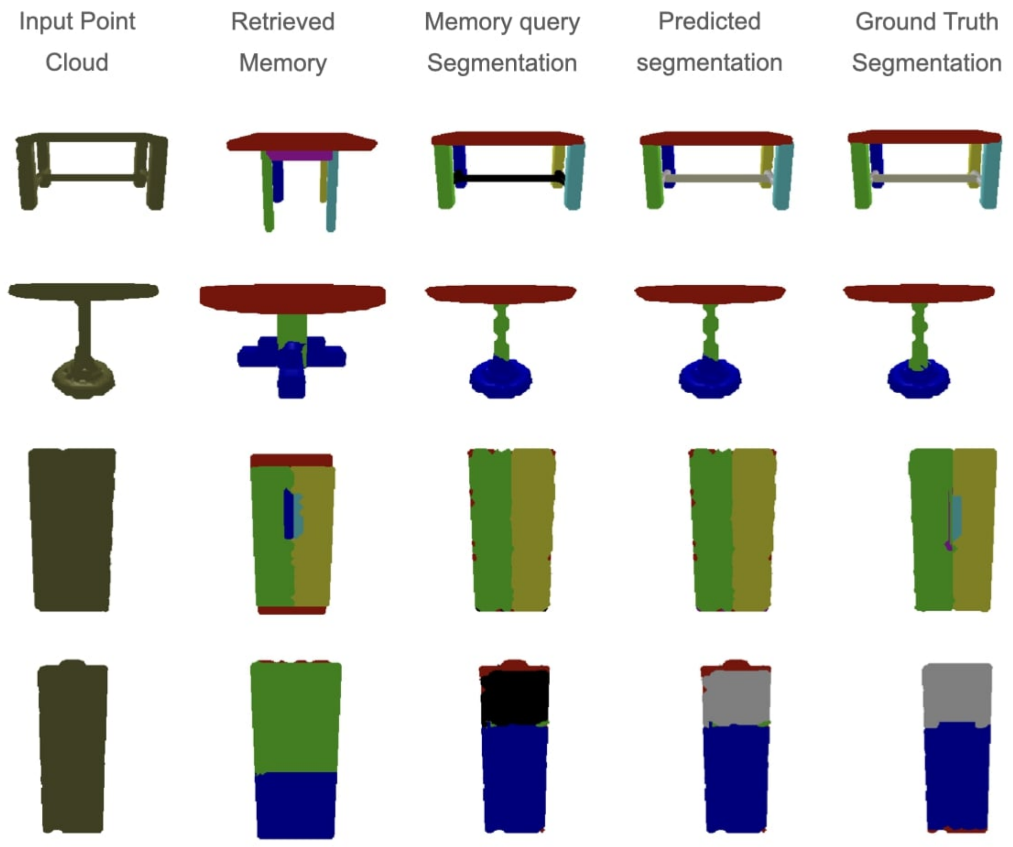


Figure C.5: Qualitative results on novel category samples from PartNet dataset [243] using Analogical Networks **without fine-tuning**.

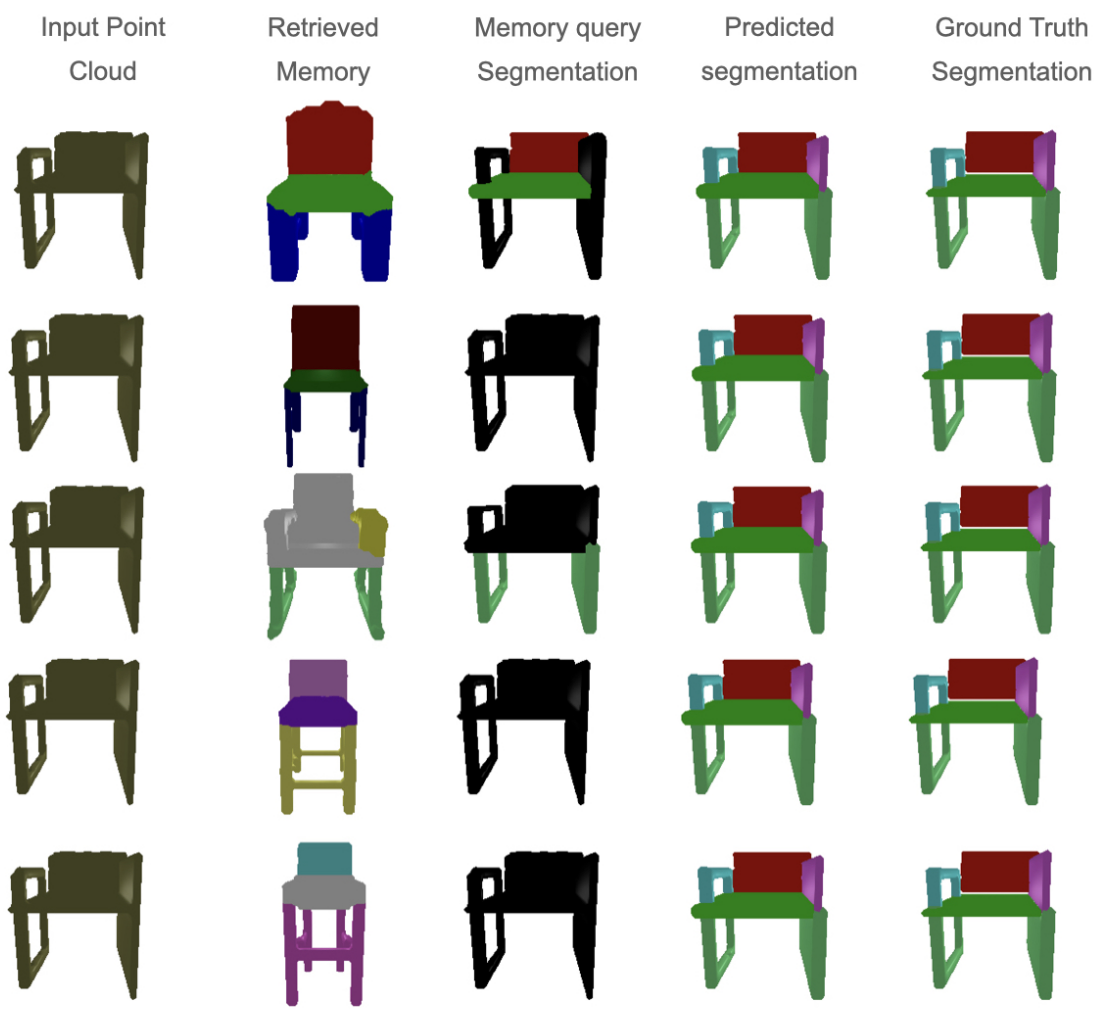


Figure C.6: Modulation using multi-memory Analogical Networks. Our model takes as input 5 different memories simultaneously and then parses the object. Each row shows the effect of a different memory. All memories decode simultaneously and we show which part each one decodes in the third column. In the fourth column we show the combined predictions of all memories and scene-agnostic queries.

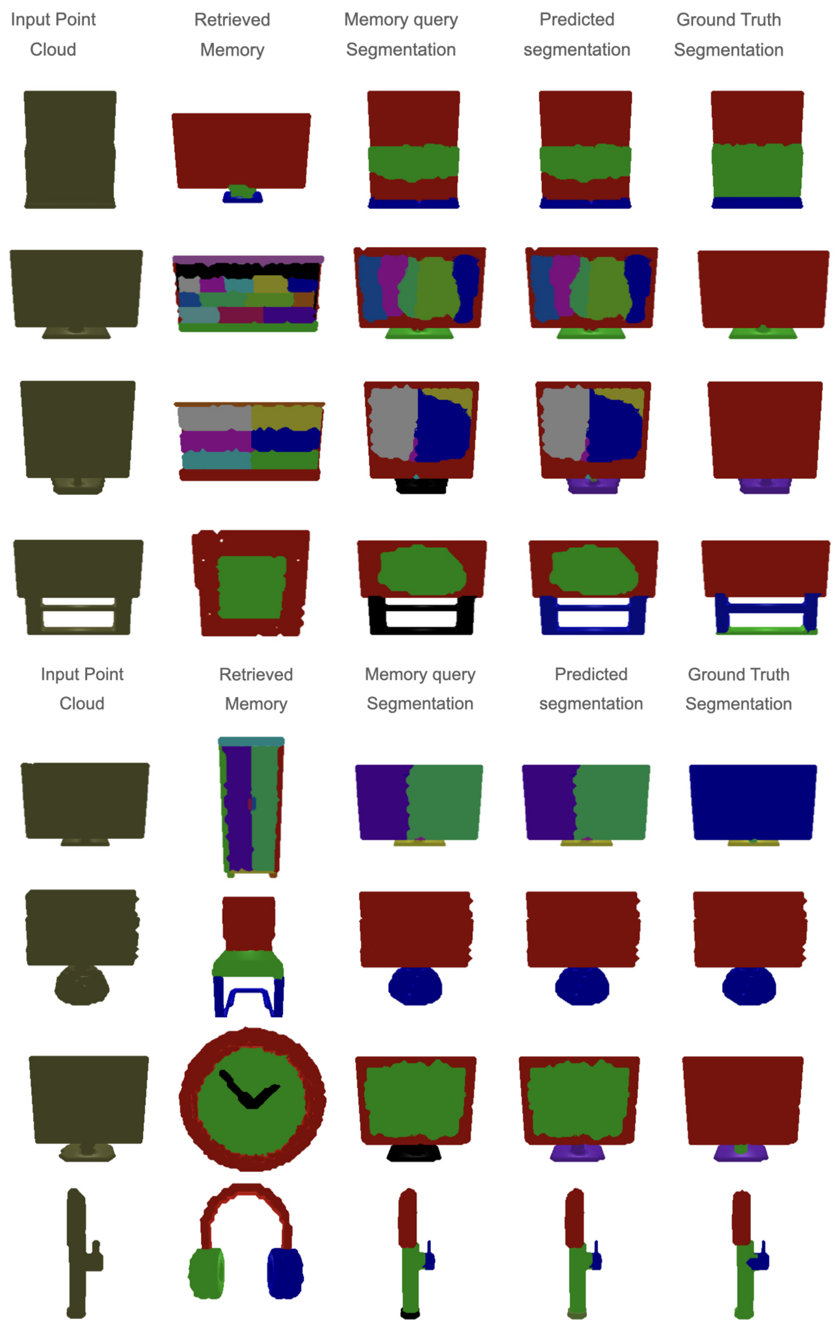


Figure C.7: To qualitatively evaluate the effect of modulation in parsing: **Top**: we manually annotate the retrieved memory with random labels that do not correspond to any PartNet level. Our model adapts to the new label space. **Bottom**: we use as memory an object of a different category. The model is able to generalize geometric correspondences across instances of totally different classes, e.g. display and clock.

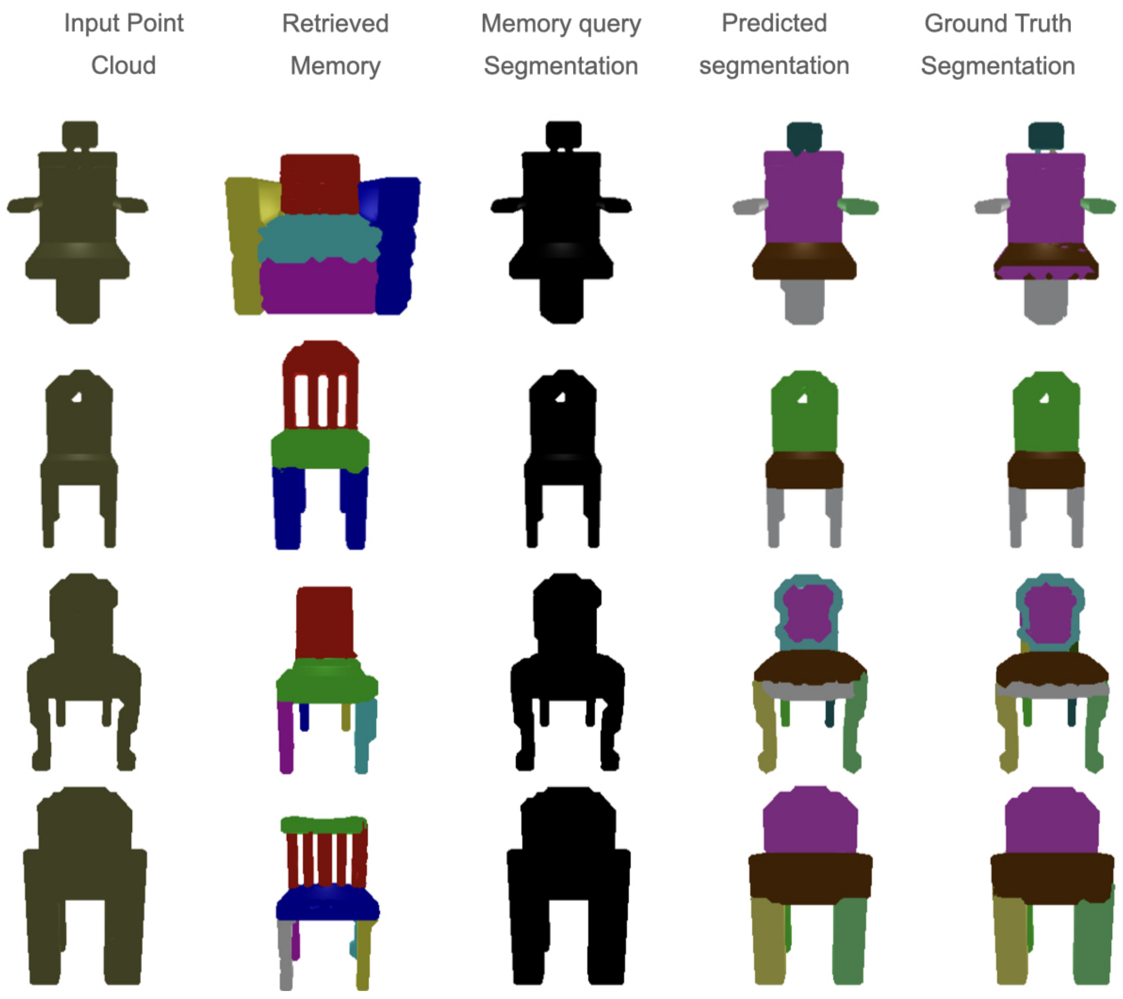


Figure C.8: We show the parsing of input point cloud using `AnalogicalNets single-mem w/o within-scene`. Most regions are black in column 3, denoting that memory part queries do not decode anything and everything is being decoded by scene-agnostic queries. This highlights the role of within-scene pre-training for the emergence of part correspondence.

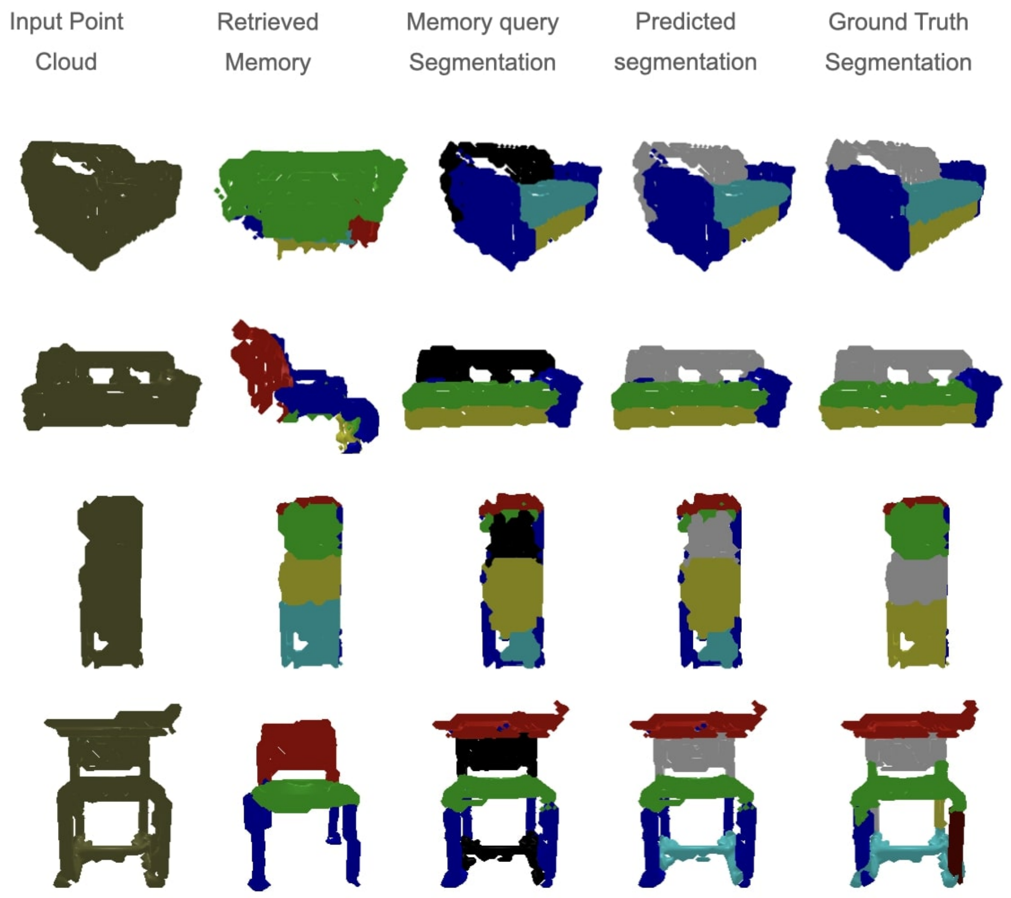


Figure C.9: Results on base category samples from ScanObjectNN [341] using Analogical Networks.

Appendix D

Energy-based Models are Zero-Shot Planners for Compositional Scene Rearrangement

In Section D.1 we give implementation details for the components of our method; in Section D.2 we present in more detail the evaluation metrics for the newly-introduced tasks; in Section D.3 we show the effectiveness of closed-loop execution for predicting the success or failure of execution and present a detailed error analysis; we visualize the learned energy landscapes in Section D.4; in Section D.5 we include additional related work on constraint-guided layout optimization.

D.1 Implementation Details

Energy-based Models: The architectures of the BinaryEBM and MultiAryEBM are shown in Figure D.1a and b respectively. We train a separate network for each concept in our library using the same demos that are used to train the other modules. We augment and repeat the samples multiple times to create an artificially larger dataset during training. We use Adam [171] optimizer, learning rate $1e-4$, batch size 128.

Buffer: During training, we fill a buffer of previously generated examples, following [70]. The buffer is updated after each training iteration to store at most 100000

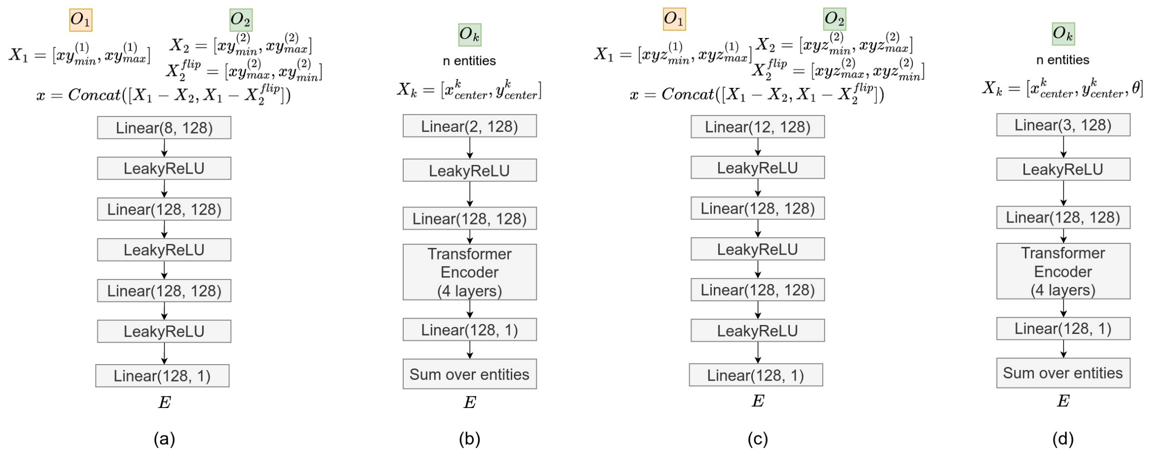

O_1
O_2
$X_1 = [xy^{(1)}_{min}, xy^{(1)}_{max}]$
$X_2 = [xy^{(2)}_{min}, xy^{(2)}_{max}]$
$X_2^{flip} = [xy^{(2)}_{max}, xy^{(2)}_{min}]$
$x = Concat([X_1 - X_2, X_1 - X_2^{flip}])$
Linear(8, 128)
LeakyReLU
Linear(128, 128)
LeakyReLU
Linear(128, 128)
LeakyReLU
Linear(128, 1)
E
(a)
O_k
n entities
$X_k = [x^k_{center}, y^k_{center}]$
Linear(2, 128)
LeakyReLU
Linear(128, 128)
Transformer Encoder (4 layers)
Linear(128, 1)
Sum over entities
E
(b)
O_1
O_2
$X_1 = [xyz^{(1)}_{min}, xyz^{(1)}_{max}]$
$X_2 = [xyz^{(2)}_{min}, xyz^{(2)}_{max}]$
$X_2^{flip} = [xyz^{(2)}_{max}, xyz^{(2)}_{min}]$
$x = Concat([X_1 - X_2, X_1 - X_2^{flip}])$
Linear(12, 128)
LeakyReLU
Linear(128, 128)
LeakyReLU
Linear(128, 128)
LeakyReLU
Linear(128, 1)
E
(c)
O_k
n entities
$X_k = [x^k_{center}, y^k_{center}, \theta]$
Linear(3, 128)
LeakyReLU
Linear(128, 128)
Transformer Encoder (4 layers)
Linear(128, 1)
Sum over entities
E
(d)

Figure D.1: **(a)**: Architecture of the EBM used for binary concepts such as “right of”. The inputs are two boxes O_1 and O_2 and the output is the energy of their relative placement. **(b)**: Architecture of the EBM used for multi-ary concepts such as “circle”. The input is a set of n entities $O_k, k = 1, \dots, n$. The output is the energy of this set of entities wrt the concept. **(c)**: Architecture of the EBM used for 3D binary concepts such as “on”. Each object is now represented by a 3D bounding box. **(d)**: Architecture of the EBM used for concepts that involve pose optimization (rotation). Each object is represented with its center and rotation wrt the global coordinate frame.

generated examples. When the buffer is full, we randomly replace older examples with incoming new ones. We initialize x_0 for Equation 5.1 by sampling from the buffer 70% of the times or loading from the data loader 30% of the times.

Regularization losses: We use the KL-loss from [70], $\mathcal{L_{KL}} = \mathbb{E}_{x^- \sim p_\theta} \bar{E}_\theta(x^-)$, where the bar on top of $\bar{E}$ indicates the stop-gradient operation (we only backpropagate to E through x^-). We additionally use the L2 energy regularization loss $\mathbb{E}_{x^+ \sim p_D} E^2_\theta(x^+) + \mathbb{E}_{x^- \sim p_\theta} E^2_\theta(x^-)$. We refer the reader to [70] for more explanation on these loss terms.

Extension to tasks with 3D information or pose: An important design choice is what parameters of the input we should be able to edit. We inject the prior knowledge that on our manipulation domain the objects move without deformations, so we fix their sizes and update only their positions. Our EBMs operate on boxes so that they can abstract relative placement without any need for object class or shape information. However, EBMs can be easily extended to optimize other types of representations, such as 3D bounding boxes or pose.

Table D.1: All operations in the domain-specific language for SREM

Operation	**Signature**	**Semantics**
Filter	(ObjectSet, ObjectConcept) → ObjectSet	Filter out set of objects based on some *Object Concept* like object name (eg. cube) or property (color, material)
BinaryEBM	(Object A, Object B, Relation) → (Pick locations, Place locations)	Executes *BinaryEBMs* for rearranging Object A and Object B to satisfy the given binary *relation* (like left of/right of/inside etc.)
MultiAryEBM	(ObjectSet, Shape Type, Property) → (Pick locations, Place locations)	Executes *MultiAryEBMs* for the given *Shape Type* (circle, line, etc.) with specified *Properties* (like size, position etc.) on a set of given objects and generates pick and place locations to complete the shape.

We train EBMs that optimize over 3D locations for relative placement. The architecture is shown in Figure D.1c. We adapt the BinaryEBM to represent boxes as (xyz_{min}, xyz_{max}) and then compute the relative representations as in the 2D case. We optimize for one 3D relation, "on". For pose-aware EBMs, we adapt the MultiAryEBM to represent objects as $(x_{center}, y_{center}, \theta)$, where θ is the rotation wrt the world frame. We then simply change the first linear layer of the MultiAryEBM to map the new input tuple to a 128-d feature vector. The architecture is shown in Figure D.1d. We show qualitative results that compose these EBMs into new concepts on our website.

Domain-Specific Language: We design a Domain-Specific Language (DSL) which extends the DSL of NS-CL [233] (designed for visual question answering in CLEVR [152]) to further predict scene generations, e.g. *"put all brown shoes in the green box"*. Detailed description of our DSL can be found in Table D.1.

Semantic Parser We construct program annotations for the language instructions of the training demos by mapping them to our DSL (rule-based). We then train our parser on all instructions using Adam optimizer with learning rate $1e-3$ and batch size 32. The same parser weights are used across all tasks. For compositional tasks, we train the parser on the descriptions from the demos that are used to finetune our baselines. The parser is the only part of our framework that needs to be updated to handle longer instructions.

Visual-language Grounder: We finetune BUTD-DETR [135] on the scenes of the training demos in simulation. BEAUTY-DETR is an encoder-decoder Detection Transformer that takes as input an image and a referential language expression and maps word spans to image regions (bounding boxes). The original BEAUTY-DETR implementation uses an additional box stream of object proposals generated by an off-the-shelf object detector. We use the variant without this box stream for simplicity. BEAUTY-DETR has been trained on real-world images. We finetune it using the weights and hyperparameters from the publicly available code of [135].

Short-term Manipulation Skills Our low-level policy network is based on Transporter Networks. Transporter Networks decompose a given task into a sequence of pick-and-place actions. Given an overhead image, the model predicts a pick location and then conditions on it to predict a place location and gripper pose. The original implementation of Transporter Networks supports training with batch size 1 only. We implement a batch-supporting version and find it more stable. We use batch size 8 and follow the original paper in other hyperparameter values.

D.2 Benchmark Generation

We extend the Ravens [394] benchmark for spatial reasoning in the PyBullet simulator. For each benchmark, we write a template sentence (e.g. “Arrange OBJ1 into a circle”) and then randomly select valid objects and colors from a pre-defined list. To test generalization, we include novel colors or novel objects in the evaluation set. Once the sentence is generated, we programatically define valid regions which satisfy the relation and then sample empty locations from it to specify object goal locations. We start by placing all objects randomly in the scene. Then, an oracle hand-designed policy picks and places the objects to the desired locations and returns a demo trajectory which consists of raw RGB-D images and pick-and-place locations. These can be used then to train a behaviour cloning policy similar to CLIPort.

Evaluation Metrics for Rearrangement Tasks To evaluate make-a-circle task, we fit a best-fit circle for the final configuration predicted by SREM. To do this, we consider the centers of the bounding boxes as points. Then, we compute the centroid of those points and the distance of each point from the centroid. This is an estimate of "radius". We compute the standard deviation of this radius. If this is lower than 0.03, then we assign a perfect reward. The reward linearly decreases when the std increases from 0.03 to 0.06. Beyond that, we give zero reward. We tuned these thresholds empirically by generating and distorting circle configurations.

We follow a similar evaluation strategy for make-a-line. Here we compute the average slope and fit a line to our data. Then we measure the standard deviation of the distance of each point from the line. We found that the same thresholds we use for circles work well for lines as well.

D.3 Additional Experiments

Details on Generalization Experiments: We conduct controlled studies of our model's generalization across three axes: a) Novel Colors b) Novel background color of the table c) Novel Objects. In each of these settings, we only change one attribute (i.e. object color, background color or object instance) while keeping everything else constant.

- *Novel colors*: We train the models with ["blue", "red", "green", "yellow", "brown", "gray", "cyan"] colors and evaluate them with unseen ["orange", "purple", "pink", "white"].
- *Novel background colors*: All models are trained with black colored tables and evaluated with randomly sampled RGB color for each instruction.
- *Novel objects*: We train the models on ["ring", "cube", "cylinder", "bowl"] and evaluate them with ["triangle", "square", "plus", "diamond", "pentagon", "rectangle", "flower", "star", "circle", "hexagon", "heart"].
- *Real-World Experiments* In our real-world experiments, we also use novel objects or novel object descriptions like "bananas", "strawberry", "small objects" which the model hasn't seen during training in simulation.

Closed-Loop Execution: As we show in Table 5.3, adding feedback roughly adds 3.5% performance boost on comp-group benchmark and 1% boost on comp-one-

Table D.2: **Performance of failure detection using our generated goal.** We check whether the boxes of the objects in the rearranged scene overlap with the locations the EBM generated. If not, we mark the rearrangement as failed (not satisfying the goal).

Benchmark	Precision	Recall	Accuracy
Comp-group	80.5	82.5	85.0
Comp-one-step	71.4	19.2	77.0

step benchmark. Retrying cannot always recover from failure, however, it would be still important to know if the execution failed so that we can request help from a human. Towards this, we evaluate the failure detection capabilities of our feedback mechanism in Table-D.2. We observe that all metrics are high for comp-group benchmark. For comp-one-step, however, the recall is very low i.e. only 19.2%. This is because in this benchmark, we only need to move one object to complete the task and hence most failures are due to wrong goal generation. Thus, the failure classifier classifies those demos as success, because indeed the model managed to achieve its predicted goal and hence results in low recall. This motivates the use of external failure classifiers in conjunction with internal goal-checking classifiers. In contrast, comp-group benchmark is harder because it requires the model to move many objects and thus results in higher chances of robot failures. These failures can be better detected and fixed by our goal-checking classifier.

Also, note that while previous works like InnerMonologue [126] show large improvements by adding closed-loop feedback, the improvements for us are smaller. This is because, by design, our model is more likely to reach its goal - since the EBM generates a visual goal and then the low-level policy predicts a pick-and-place location **within** the predicted goal, it is very likely to satisfy its predicted goal. Indeed, in the comp-one-step, the model satisfies its goal in 93% cases and in 70% cases in the comp-group benchmark. In contrast, InnerMonologue does not have any built-in mechanisms for promoting goal-reaching behaviours and thus the difference in performance with additional goal-satisfying constraints is larger for that method.

There is a lot of potential in designing better goal checkers. As already discussed, we can incorporate external success classifiers like those used by prior literature [126] in conjunction with goal-checking classifiers. Explicit goal generation allows then to

Table D.3: **Error Analysis of SREM on the benchmark comp-group-seen-colors.**

Error Mode	Error Percentage
Robot failure	6.0
Goal Generation failure	11.7
Grounding failure	5.1
Language Parsing failure	0.0

check validity of goals directly even before actual robot execution (which makes it safer and less expensive). We leave this for future work. Another promising direction is to add object trackers in the feedback mechanism to keep track of the objects and detect if a failure happened. This is important, if we have multiple objects that are visually similar and hence we would need to keep track of which object corresponds to which goal and retry if it failed to reach it.

Error Analysis: We conduct a detailed error analysis of our model on comp-group benchmark, shown in Table-D.3. We find that robot failures, i.e. collisions or failure to pick/place objects, result in 6.0% drop in accuracy. Goal generation adds 11.7% to the failure. Visual grounding leads to 5.1% errors while we find language parsing to be nearly perfect.

D.4 EBM Energy Landscape Visualization

We visualize the energy landscape of our EBMs for different concepts in Figure-D.2. For binary relations (A, rel, B), we fix B's bounding box in the scene and then move the bounding box of A all over the scene and evaluate the energy of the configuration at each position. For shapes, it is impossible to represent the landscape in 2D or 3D because we need to jointly consider all possible combinations of objects in the scene. Hence, we fix all but one object in a valid circle/line location and move a free box in all possible regions. For compositions of relations, we move only one box and score the sum of energy for all constraints. We expect the energy to be low in regions which satisfy the described concept (relation(s) or shape) and high elsewhere. We observe that the energy landscape is usually smooth with low values in valid regions and high otherwise.

D.5 Additional Related Work

Constraint-Guided Layout Optimization: Automatic optimization for object rearrangement has been studied outside the field of robotics. [387] and [237] use few user-annotated examples of scenes to adapt the hyperparameters of task-specific cost functions, which are then minimized using standard optimization algorithms (hill climbing and/or simulated annealing). To learn those hyperparameters from data, these approaches fit statistical models, e.g. Mixtures of Gaussian, to the given samples. [79] further employ such optimization constraints into an interactive environment, where the user can provide an initial layout and the algorithm suggests improvements. All these approaches require expert knowledge to manually design rules and cost function, namely [387] identifies seven and [79] eleven expert-suggested criteria for successful rearrangement. Since they are hand-crafted, these methods do not generalize beyond the domain of furniture arrangement. In contrast, energy optimization is purely data-driven and domain-agnostic: a neural network scores layouts, assigning high energy to those that do not satisfy the (implicit) constraints and low energy to those who do, essentially modeling the underlying distribution of valid layouts.

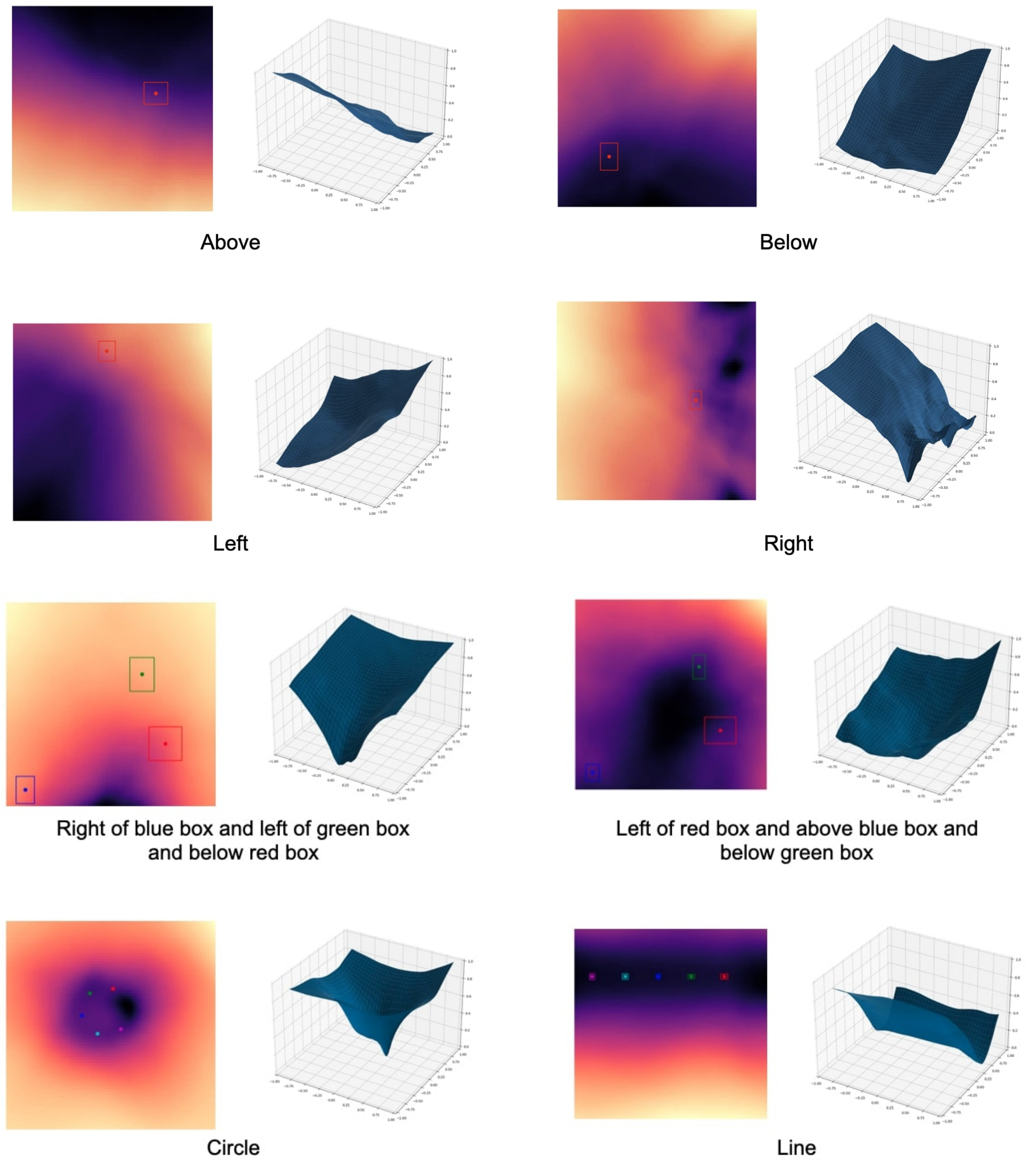


Figure D.2: **EBM Energy Landscape visualization:** The boxes shown here remain fixed and we score the energy by moving another box all along the workspace. Energy decreases from white to orange to purple to black.

Appendix E

Act3D: 3D Feature Field Transformers for Multi-Task Robotic Manipulation

E.1 Real-world Setup

Our real-robot setup contains a Franka Panda robotic arm equipped with a parallel jaw gripper, as shown in Figure E.1. We get RGB-D input from a single Azure Kinect sensor at a front view at 30Hz. The image input is of resolution 1280×720, we crop and downsample it to 256×256. We calibrate the extrinsics of the camera with respect to the robot base using the `easy_handeye` ROS package. We extract keyposes from demonstrations in the same was as in simulation. Our real-world multi-task policy is trained on 4 V100 GPUs for 3 days, and we run inference on a desk-

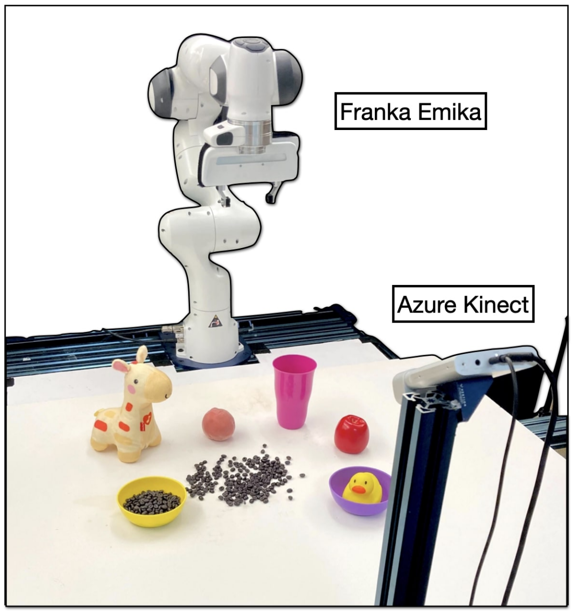


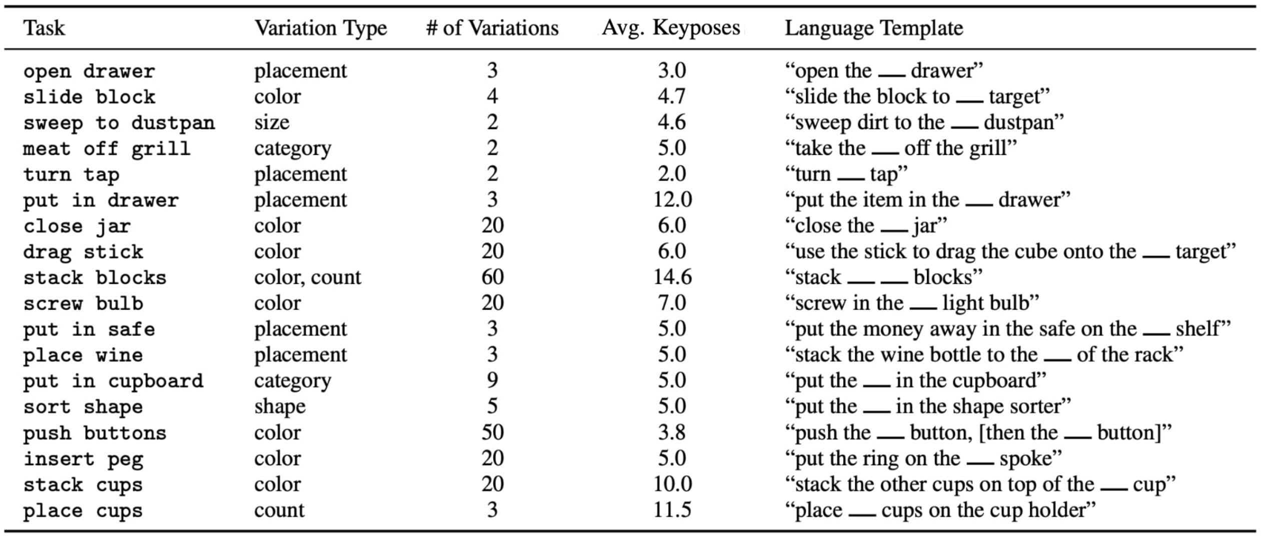
Figure E.1: **Real-world setup.**

https://github.com/IFL-CAMP/easy_handeye top with a single RTX4090 GPU. For robot control, we use the open-source `frankapy` package to send real-time position-control commands to the robot.

E.2 RLBench Simulation Setup

To ensure fair comparison with prior work, we use $n_{\text{cam}} \in \{3, 4\}$ cameras for simulated experiments depending on the evaluation setting. In our single-task evaluation setting first proposed by HiveFormer [103], we use the same 3 cameras they do $\{O_{\text{left}}, O_{\text{right}}, O_{\text{wrist}}\}$. In our multi-task evaluation setting first proposed by PerAct [312], we use the same 4 cameras they do $\{O_{\text{front}}, O_{\text{left}}, O_{\text{right}}, O_{\text{wrist}}\}$.

E.3 RLBench Tasks



Task	Variation Type	# of Variations	Avg. Keyposes	Language Template
`open drawer`	placement	3	3.0	"open the ___ drawer"
`slide block`	color	4	4.7	"slide the block to ___ target"
`sweep to dustpan`	size	2	4.6	"sweep dirt to the ___ dustpan"
`meat off grill`	category	2	5.0	"take the ___ off the grill"
`turn tap`	placement	2	2.0	"turn ___ tap"
`put in drawer`	placement	3	12.0	"put the item in the ___ drawer"
`close jar`	color	20	6.0	"close the ___ jar"
`drag stick`	color	20	6.0	"use the stick to drag the cube onto the ___ target"
`stack blocks`	color, count	60	14.6	"stack ___ ___ blocks"
`screw bulb`	color	20	7.0	"screw in the ___ light bulb"
`put in safe`	placement	3	5.0	"put the money away in the safe on the ___ shelf"
`place wine`	placement	3	5.0	"stack the wine bottle to the ___ of the rack"
`put in cupboard`	category	9	5.0	"put the ___ in the cupboard"
`sort shape`	shape	5	5.0	"put the ___ in the shape sorter"
`push buttons`	color	50	3.8	"push the ___ button, [then the ___ button]"
`insert peg`	color	20	5.0	"put the ring on the ___ spoke"
`stack cups`	color	20	10.0	"stack the other cups on top of the ___ cup"
`place cups`	count	3	11.5	"place ___ cups on the cup holder"

Figure E.2: **PerAct [312] tasks.** We adopt the multi-task multi-variation setting from PerAct [312] with 18 tasks and 249 unique variations across object placement, color, size, category, count, and shape.

https://github.com/iamlab-cmu/frankapy

We adapt the single-task setting of HiveFormer [103] with 74 tasks grouped into 9 categories according to their key challenges. The 9 task groups are defined as follows:

- The **Planning** group contains tasks with multiple sub-goals (e.g. picking a basket ball and then throwing the ball). The included tasks are: basketball in hoop, put rubbish in bin, meat off grill, meat on grill, change channel, tv on, tower3, push buttons, stack wine.
- The **Tools** group is a special case of planning where a robot must grasp an object to interact with the target object. The included tasks are: slide block to target, reach and drag, take frame off hanger, water plants, hang frame on hanger, scoop with spatula, place hanger on rack, move hanger, sweep to dustpan, take plate off colored dish rack, screw nail.
- The **Long term** group requires more than 10 macro-steps to be completed. The included tasks are: wipe desk, stack blocks, take shoes out of box, slide cabinet open and place cups.
- The **Rotation-invariant** group can be solved without changes in the gripper rotation. The included tasks are: reach target, push button, lamp on, lamp off, push buttons, pick and lift, take lid off saucepan.
- The **Motion planner** group requires precise grasping. As observed in [81] such tasks often fail due to the motion planner. The included tasks are: toilet seat down, close laptop lid, open box, open drawer, close drawer, close box, phone on base, toilet seat up, put books on bookshelf.
- The **Multimodal** group can have multiple possible trajectories to solve a task due to a large affordance area of the target object (e.g. the edge of a cup). The included tasks are: pick up cup, turn tap, lift numbered block, beat the buzz, stack cups.
- The **Precision** group involves precise object manipulation. The included tasks are: take usb out of computer, play jenga, insert onto square peg, take umbrella out of umbrella stand, insert usb in computer, straighten rope, pick and lift small, put knife on chopping board, place shape in shape sorter, take toilet roll off stand, put umbrella in umbrella stand, setup checkers.
- The **Screw** group requires screwing an object. The included tasks are: turn

oven on, change clock, open window, open wine bottle.

- The **Visual Occlusion** group involves tasks with large objects and thus there are occlusions from certain views. The included tasks are: close microwave, close fridge, close grill, open grill, unplug charger, press switch, take money out safe, open microwave, put money in safe, open door, close door, open fridge, open oven, plug charger in power supply

E.4 Further Architecture Details

Relative 3D cross-attentions We featurize each of the 3D ghost points and a parametric query (used to select via inner-product one of the ghost points as the next best end-effector position in the decoder) independently through cross-attentions to the multi-scale 3D scene feature cloud, language tokens, and proprioception. Featurizing ghost points independently, without self-attentions to one another, enables sampling more ghost points at inference time to improve performance, as we show in Section 6.4. Our cross-attentions use relative 3D position information and are implemented efficiently with rotary positional embeddings [325].

Given a point $\mathbf{p} = (x, y, z) \in \mathbb{R}^3$ and its feature $\mathbf{x} \in \mathbb{R}^d$, the rotary position encoding function $\mathbf{PE}$ is defined as:

$$\mathbf{PE}(\mathbf{p}, \mathbf{x}) = \mathbf{M}(\mathbf{p})\mathbf{x} = \begin{bmatrix} \mathbf{M}_1 & & \\ & \ddots & \\ & & \mathbf{M}_{d/6} \end{bmatrix} \mathbf{x} \tag{E.1}$$

$$\mathbf{M}_k = \begin{bmatrix} \cos x\theta_k & -\sin x\theta_k & 0 & 0 & 0 & 0 \\ \sin x\theta_k & \cos x\theta_k & 0 & 0 & 0 & 0 \\ 0 & 0 & \cos y\theta_k & -\sin y\theta_k & 0 & 0 \\ 0 & 0 & \sin y\theta_k & \cos y\theta_k & 0 & 0 \\ 0 & 0 & 0 & 0 & \cos z\theta_k & -\sin z\theta_k \\ 0 & 0 & 0 & 0 & \sin z\theta_k & \cos z\theta_k \end{bmatrix} \tag{E.2}$$

where $\theta_k = \frac{1}{10000^{6(k-1)/d}}$, $k \in \{1, .., d/6\}$. The dot product of two positionally encoded features is

$$\mathbf{PE}(\mathbf{p}_i, \mathbf{x}_i)^T \mathbf{PE}(\mathbf{p}_j, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{M}(\mathbf{p}_i)^T \mathbf{M}(\mathbf{p}_j) \mathbf{x}_j = \mathbf{x}_i^T \mathbf{M}(\mathbf{p}_j - \mathbf{p}_i) \mathbf{x}_j \tag{E.3}$$

which depends only on the relative positions of points $\mathbf{p}_i$ and $\mathbf{p}_j$.

We extract two feature maps per 256x256 input image view: 32x32 coarse visual tokens and 128x128 fine visual tokens. We use three ghost point sampling stages: first uniformly across the entire workspace (roughly 1 meter cube), then uniformly in a 16 centimeter diameter ball, and finally in a 4 centimeter diameter ball. The coarsest ghost points attend to a global coarse scene feature cloud (32x32xn_{cam} coarse visual tokens) whereas finer ghost points attend to a local fine scene feature cloud (the closest 32x32xn_{cam} out of the total 128x128xn_{cam} fine visual tokens). During training, we sample 1000 ghost points in total split equally across the three stages. At inference time, we can trade-off extra prediction precision and task performance for additional compute by sampling more ghost points than the model ever saw at training time ($10,000$ in our experiments). We show in ablations in Section 6.4 that our framework is robust to these hyper-parameters but tying weights across sampling stages and relative 3D cross-attention are both crucial for generalization. We use 2 layers of cross-attention and an embedding size 60 for single-task experiments and 120 for multi-task experiments. Training samples are augmented with random crops of RGB-D images and ± 45.0 yaw rotation perturbations (only in the real world as this degrades performance in simulation as we show in Section 6.4). The cropping operation is performed on aligned RGB and depth frames together, thus maintain pixel-level correspondence. We use a batch size 16 on a NVIDIA 32GB V100 GPU for 200k steps (one day) for single-task experiments, and a batch size 48 on 8 Nvidia 32GB V100 GPUs for 600K steps (5 days) for language-conditioned multi-task experiments. At test time, we call a low-level motion planner to reach predicted keyposes. In simulation, we use native motion planner implementation provided in RLBench, which is a sampling-based BiRRT [176] motion planner powered by Open Motion Planning Library (OMPL) [327] under the hood. For real-world experiments, we use the same BiRRT planner provided by the MoveIt! ROS package [44].

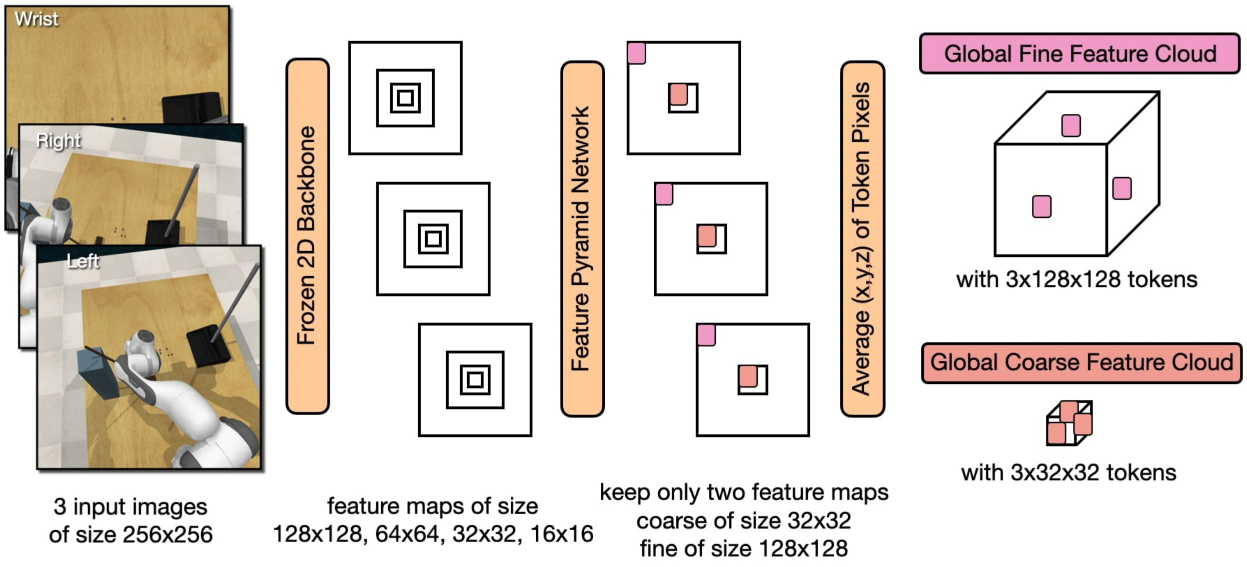

Wrist
Right
Left
3 input images of size 256x256
Frozen 2D Backbone
feature maps of size 128x128, 64x64, 32x32, 16x16
Feature Pyramid Network
keep only two feature maps coarse of size 32x32 fine of size 128x128
Average (x,y,z) of Token Pixels
Global Fine Feature Cloud
with 3x128x128 tokens
Global Coarse Feature Cloud
with 3x32x32 tokens

Figure E.3: **Scene Feature Cloud Generation**. We encode each image independently with a pre-trained and frozen vision backbone to get multi-scale feature maps, pass these feature maps through a feature pyramid network and retain only two: a coarse feature map (at a granularity that lets ghost points attend to all tokens within GPU memory) and a fine feature map (as spatially precise as afforded by input images and the backbone). We lift visual tokens from these two feature maps for each image to 3D scene feature clouds by averaging the positions of pixels in each 2D visual token.

E.5 High Precision Experiments

In this section, we further investigate the ability of Act3D to improve over existing 3D methods that voxelize the workspace for high-precision tasks. We compare two variants of Act3D against PerAct [312] on three high-precision tasks in success rate. The first Act3D variant is the standard architecture used in the remainder of our experiments operating on 256x256 input image views; the second operates on higher resolution 512x512 input image views, from which it extracts four times as many visual tokens with more precise 3D positions. This further tests the ability of Act3D to provide high precision by processing higher-resolution RGB-D views at the cost of extra compute.

Act3D improves over PerAct on high precision tasks and can further benefit from higher resolution RGB-D images, at the cost of extra compute.

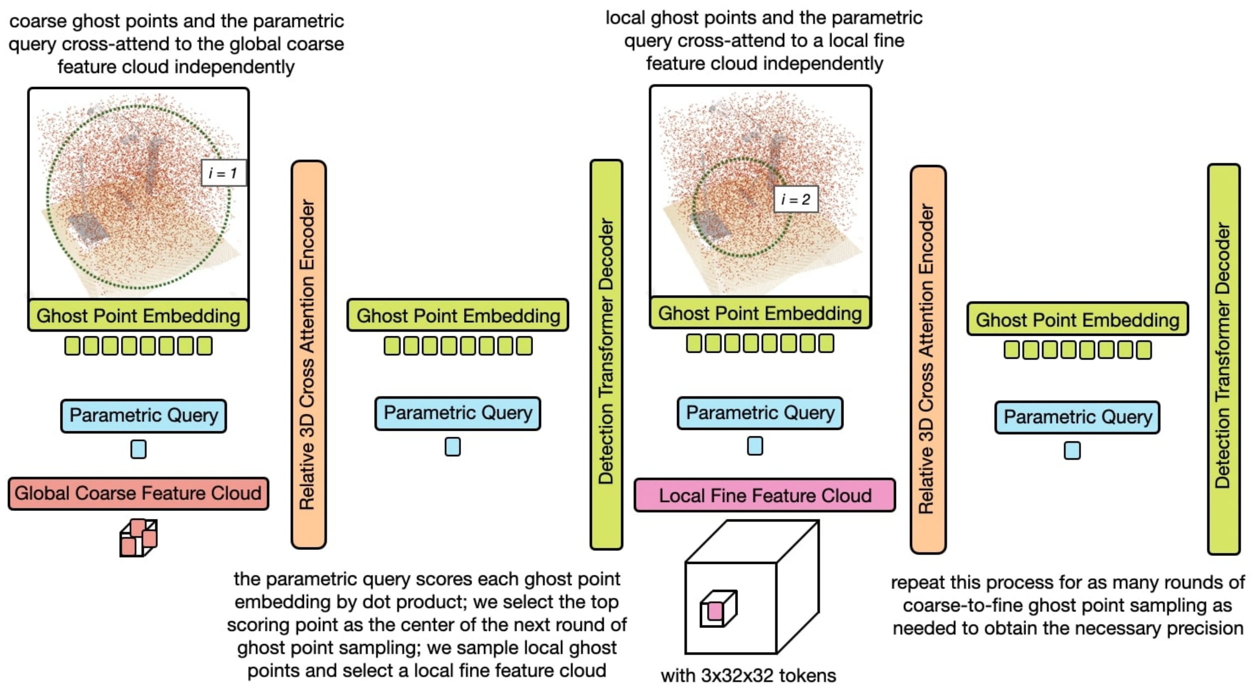

coarse ghost points and the parametric query cross-attend to the global coarse feature cloud independently
i = 1
Ghost Point Embedding
Parametric Query
Global Coarse Feature Cloud
Relative 3D Cross Attention Encoder
Ghost Point Embedding
Parametric Query
Detection Transformer Decoder
the parametric query scores each ghost point embedding by dot product; we select the top scoring point as the center of the next round of ghost point sampling; we sample local ghost points and select a local fine feature cloud
local ghost points and the parametric query cross-attend to a local fine feature cloud independently
i = 2
Ghost Point Embedding
Parametric Query
Local Fine Feature Cloud
with 3x32x32 tokens
Relative 3D Cross Attention Encoder
Ghost Point Embedding
Parametric Query
Detection Transformer Decoder
repeat this process for as many rounds of coarse-to-fine ghost point sampling as needed to obtain the necessary precision

Figure E.4: **Iterative Ghost Point Sampling, Featurization, and Selection**.

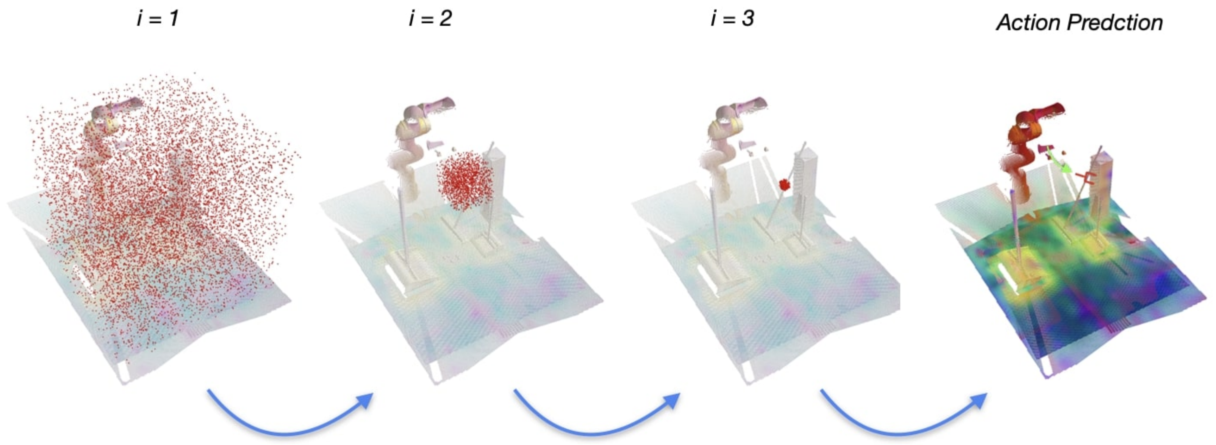


Figure E.5: **Iterative Ghost Point Sampling, Featurization, and Selection**.

E.6 Further ablations

Augmentations: Random crops of RGB-D images boost success rate by 6.5%, but yaw rotation perturbations drop it by 11.9%. This is in line with PerAct [312] results in RLBench.

Method	insert peg	sort shape	screw nail
PerAct	16	31	12
Act3D (256x256)	29	34	31
Act3D (512x512)	**47**	**43**	**55**

Table E.1: **Ablations.**

	Model	Average success rate in single-task setting (5 tasks)
Core design choices	Best Act3D model (evaluated in Fig. 6.3)	**98.1**
	Only 2 stages of coarse-to-fine sampling: full workspace, 16 cm ball, regress an offset	93.6
	No weight tying across stages	80.6
	Absolute 3D positional embeddings	55.4
	Attention to only global coarse visual features	89.8
	Only 1000 ghost points at inference time	93.2
Viewpoint changes	Best Act3D model (evaluated in Fig. 6.3)	**74.2**
	HiveFormer	20.4
Augmentations	No image augmentations	**91.6**
	With rotation augmentations	86.2
Hyperparameter sensitivity	Double sampling ball diameters: 32 cm and 8 cm	96.6
	Halve sampling ball diameters: 8 cm and 2 cm	91.2
	500 ghost points at training time	95.8
	2000 ghost points at training time (need 2 GPUs)	**98.4**
		Multi-task setting (18 tasks)
Backbone	CLIP ResNet50 backbone	**65.1**
	ImageNet ResNet50 backbone	53.4
	No backbone (raw RGB)	45.2

Hyperparameter sensitivity: Act3D is robust to variations in hyperparameters. Doubling the diameter of ghost point sampling balls from (16 cm, 4 cm) to (32 cm, 8 cm) drops success rate by 1.5% and halving it to (8 cm, 2 cm) by 6.9%. Halving the total number of ghost points sampled from 1,000 to 500 drops success rate by 2.3% whereas doubling it to 2,000 increases success rate by 0.3%. We use 1,000 ghost points in our experiments to allow training with a single GPU per task.

Appendix F

ChainedDiffuser: Unifying Trajectory Diffusion and Keypose Prediction for Robotic Manipulation

F.1 Noise schedulers for Local Trajectory Diffuser

We model local trajectory optimization as a discrete-time diffusion process, which we implement using the DDPM sampler [118]. DDPM uses a non-parametric time-dependent noise variance scheduler β_k, which defines how much noise is added at each time step. We adopt a scaled linear schedule for the position and a squared cosine schedule for the rotation of each trajectory step.

$$x_{k-1} = \frac{\sqrt{\bar{\alpha}_{k-1}}\beta_k}{1-\bar{\alpha}_k}(x_k - \epsilon_\theta(x_k, k, \mathbf{c})) + \frac{\sqrt{\alpha_k}(1-\bar{\alpha}_{k-1})}{1-\bar{\alpha}_k}x_k + \frac{1-\bar{\alpha}_{k-1}}{1-\bar{\alpha}_k}\beta_k\mathbf{z} \tag{F.1}$$

By defining $\alpha_k = 1 - \beta_k$, and $\bar{\alpha}_k = \prod_{i=1}^{k} \alpha_i$, we can now obtain the analytical form of $\lambda_k, \gamma_k, \sigma_k$ in Equation 7.5 as follows:

$$\lambda_k = \frac{1}{\sqrt{\alpha_k}} \tag{F.2}$$

$$\gamma_k = \frac{1-\alpha_k}{\sqrt{1-\bar{\alpha}_k}} \tag{F.3}$$

$$\sigma_k = \frac{1-\bar{\alpha}_{k+1}}{1-\bar{\alpha}_k}\beta_k \tag{F.4}$$

where k is the diffusion denoising timestep.

F.2 Real-world Setup

Our real-robot setup contains a real Franka Panda robotic arm equipped with a parallel jaw gripper, as shown in Figure F.1. We use a single Azure Kinect sensor to provide RGB-D input signal from the front view at 30Hz. The image input is of resolution 1280 × 720, and we crop and downsample it to 256 × 256 before feeding it to our model. We calibrate the extrinsics of the camera with respect to the robot base using the `easy_handeye` ROS package. Our full model generates dense trajectories, thus we do not use low-level motion planners. We collect 6-DoF human demonstrations by tele-operating the robot using a SpaceMouse at 30Hz, following [38]. We use the same strategy for keyframe extraction as in simulation. Our real-world multi-task policy is trained on 4 A100 GPUs for 3 days. Inference is done on a desktop with a single RTX4090 GPU, running Ubuntu 20.04 and ROS

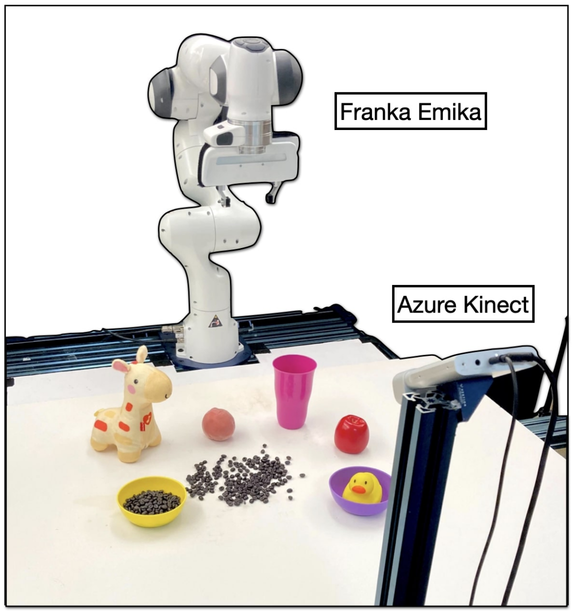


Figure F.1: Real-world setup.

https://github.com/IFL-CAMP/easy_handeye
https://3dconnexion.com/us/product/spacemouse-compact/

Table F.1: Success rate comparison with randomized camera viewpoint.

	pick & lift	pick-up cup	push button	put knife	put money	reach target	slide block	stack wine	take money	take umbrella	Mean	Δ
HiveFormer	26	74	**98**	43	63	**98**	13	45	77	85	62.2	-26.2
ChainedDiffuser	**72**	**91**	97	**78**	**88**	96	**32**	**90**	**92**	**91**	**82.7**	**-13.1**

Noetic. For robot control, we use the open-source `frankapy` package to send real-time position-control commands to the robot.

F.3 Robustness of 2D and 3D Methods under Varying Camera Viewpoints

In order to better understand how much it helps to use 3D information in Chained-Diffuser compared with prior 2D methods, we conducted experiments to evaluate the robustness of ChainedDiffuser, InstructRL and HiveFormer under varying camera viewpoint. At test time, we randomly perturb the two shoulder cameras by [20, 30] degrees, while keeping the same camera look-at point. InstructRL completely fails on all the tasks and yields unreasonable actions, as it operates purely on 2D inputs and directly regresses action outputs. It fails to handle distribution shift in image input due to changing camera viewpoint. HiveFormer uses depth image and 2.5D architecture: it selects the highest-score pixel and uses its corresponding depth to compute 3D actions, thus presenting certain robustness. We report numbers of HiveFormer and ChainedDiffuser in Table 6. Our model still performs reasonably well when the cameras are perturbed, achieving the smallest performance drop, showing desirable viewpoint-invariance. This is an advantage of reasoning directly in 3D.

https://github.com/iamlab-cmu/frankapy

Appendix G

3D Diffuser Actor: Policy Diffusion with 3D Scene Representations

G.1 Additional Experimental Results and Details

G.1.1 Robustness to noisy depth information on RLBench

We evaluate the robustness of 3D Diffuser Actor to noisy depth information. Our results are presented in Table G.1. We adopt the single-view setup on RLBench, reporting success rates on 10 tasks using the *front* camera view. We add Gaussian noise with a variance of 0.01 and 0.03 to the 3D location of each pixel. We evaluate the final checkpoints across 3 seeds with 25 episodes for each task. With mild perturbation, 3D Diffuser Actor can still predict actions precisely, achieving an average success rate of 72.4%. We only observed 6% absolute performance drop. With strong perturbation, 3D Diffuser Actor obtains an average success rate of 50.1%. Even though 3D Diffuser Actor was trained with clean depth maps, it performs reasonably well with noisy depth information.

G.1.2 Failure cases on RLBench

We analyze the failure modes of 3D Diffuser Actor on RLBench. We categorize the failure cases into 4 types: 1) pose precision, where predicted end-effector poses are

σ	Avg. Success.	close jar	open drawer	sweep to dustpan	meat off grill	turn tap	slide block	put in drawer	drag stick	push buttons	stack blocks
0 (clean)	**78.4**	**82.7**$_{\pm1.9}$	**89.3**$_{\pm7.5}$	**94.7**$_{\pm1.9}$	**88.0**$_{\pm5.7}$	80.0$_{\pm8.6}$	92.0$_{\pm0.0}$	**77.3**$_{\pm3.8}$	**98.7**$_{\pm1.9}$	69.3$_{\pm5.0}$	**12.0**$_{\pm3.7}$
0.01	72.4	80.0$_{\pm3.3}$	81.3$_{\pm1.9}$	68.0$_{\pm3.3}$	85.3$_{\pm5.0}$	**82.7**$_{\pm7.5}$	92.0$_{\pm0.0}$	65.3$_{\pm3.8}$	93.3$_{\pm3.8}$	**70.7**$_{\pm1.9}$	5.3$_{\pm1.9}$
0.03	50.1	41.3$_{\pm6.8}$	25.3$_{\pm10.0}$	45.3$_{\pm3.8}$	74.7$_{\pm3.8}$	89.3$_{\pm3.8}$	89.3$_{\pm3.8}$	18.7$_{\pm1.9}$	48.0$_{\pm3.3}$	68.0$_{\pm5.7}$	1.3$_{\pm1.9}$

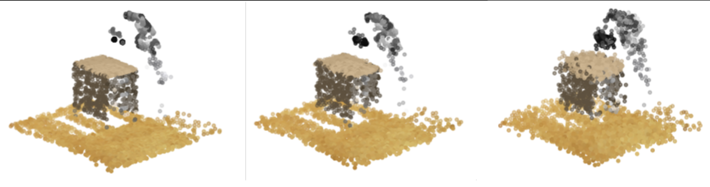

$\sigma = 0$ (clean) $\sigma = 0.01$ $\sigma = 0.03$

Table G.1: **Multi-Task performance on noisy depth information. Top:** We evaluate the robustness of 3D Diffuser Actor to noisy depth information. We adopt the single-view setup in RLBench, reporting success rates on 10 tasks using the *front* camera view. We add Gaussian noise with a variance of 0.01 and 0.03 to the 3D location of each pixel. We evaluate the final checkpoints across 3 seeds with 25 episodes for each task. 3D Diffuser Actor performs reasonably well with noisy depth information. **Bottom:** Visualization of the 3D location of each pixel with different levels of noise.

too imprecise to satisfy the success condition, 2) instruction understanding, where the policy fails to understand the language instruction to grasp the target object, 3) long-horizon task completion, where the policy fails to complete an intermediate task of a long-horizon task, and 4) path planning, where the motion planner fails to find an optimal path to reach the target end-effector pose.

We conduct the experiment with single-view setup on RLBench, aggregating the failure cases among 10 manipulation tasks. As shown in Fig. G.1, the major failure mode is imprecise prediction of end-effector poses. This failure mode often occurs in manipulation tasks that require high precision of predicted poses, such as *stack blocks*, *open drawer* and *turn tap*. Confusion of language instruction is another major failure mode. This mode often occurs in the scene with multiple objects, where the policy fails to grasp the target object specified in the instruction, such as *stack blocks*, *close jar* and *push buttons*. On RLBench, we deploy a sample-based motion planner to find the path to reach the keypose. The motion planner could sometimes fail to find the path. Lastly, 3D Diffuser Actor might fail to complete an intermediate task, such as *open drawer* of *put item in drawer* task.

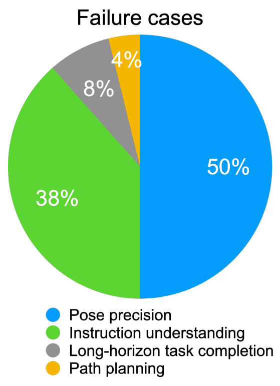


Figure G.1: **Failure cases on RLBench on the setup of GNFactor.** We categorize the failure cases into 4 types: **1)** precise pose prediction, where predicted end-effector poses are too imprecise to satisfy the success condition, **2)** instruction understanding, where the policy fails to understand the language instruction to grasp the target object, **3)** long-horizon task completion, where the policy fails to complete an intermediate task of a long-horizon task, and **4)** path planning, where the motion planner fails to find an optimal path to reach the target end-effector pose. Imprecise action prediction and confusion of language instruction are two major failure modes on RLBench.

G.1.3 RLBench tasks under multi-view setup

We provide an explanation of the RLBench tasks and their success conditions under the multi-view setup for self-completeness. All tasks vary the object pose, appearance and semantics, which are not described in the descriptions below. For more details, please refer to the PerAct paper [312].

1. Open a drawer: The cabinet has three drawers (top, middle and bottom). The agent is successful if the target drawer is opened. The task on average involves three keyposes.

2. Slide a block to a colored zone: There is one block and four zones with different colors (red, blue, pink, and yellow). The end-effector must push the block to the zone with the specified color. On average, the task involves approximately 4.7 keyposes

3. Sweep the dust into a dustpan: There are two dustpans of different sizes (short and tall). The agent needs to sweep the dirt into the specified dustpan. The task on average involves 4.6 keyposes.

4. Take the meat off the grill frame: There is chicken leg or steck. The agent needs to take the meat off the grill frame and put it on the side. The task involves 5 keyposes.

5. Turn on the water tap: The water tap has two sides of handle. The agent needs

to rotate the specified handle 90°. The task involves 2 keyposes.

6. Put a block in the drawer: The cabinet has three drawers (top, middle and bottom). There is a block on the cabinet. The agent needs to open and put the block in the target drawer. The task on average involves 12 keyposes.
7. Close a jar: There are two colored jars. The jar colors are sampled from a set of 20 colors. The agent needs to pick up the lid and screw it in the jar with the specified color. The task involves six keyposes.
8. Drag a block with the stick: There is a block, a stick and four colored zones. The zone colors are sampled from a set of 20 colors. The agent is successful if the block is dragged to the specified colored zone with the stick. The task involves six keyposes.
9. Stack blocks: There are 8 colored blocks and 1 green platform. Each four of the 8 blocks share the same color, while differ from the other. The block colors are sampled from a set of 20 colors. The agent needs to stack N blocks of the specified color on the platform. The task involves 14.6 keyposes.
10. Screw a light bulb: There are 2 light bulbs, 2 holders, and 1 lamp stand. The holder colors are sampled from a set of 20 colors. The agent needs to pick up and screw the light bulb in the specified holder. The task involves 7 keyposes.
11. Put the cash in a safe: There is a stack of cash and a safe. The safe has three layers (top, middle and bottom). The agent needs to pick up the cast and put it in the specified layer of the safe. The task involves 5 keyposes.
12. Place a wine bottle on the rack: There is a bottle of wine and a wooden rack. The rack has three slots (left, middle and right). The agent needs to pick up and place the wine at the specified location of the wooden rack. The task involves 5 keyposes.
13. Put groceries in the cardboard: There are 9 YCB objects and a cupboard. The agent needs to grab the specified object and place it in the cupboard. The task involves 5 keyposes.
14. Put a block in the shape sorter: There are 5 blocks of different shapes and a sorter with the corresponding slots. The agent needs to pick up the block with the specified shape and insert it into the slot with the same shape. The task

involves 5 keyposes.

15. Push a button: There are 3 buttons, whose colors are sampled from a set of 20 colors. The agent needs to push the colored buttons in the specified sequence. The task involves 3.8 keyposes.
16. Insert a peg: There is 1 square, and 1 spoke platform with three colored spoke. The spoke colors are sampled from a set of 20 colors. The agent needs to pick up the square and put it onto the spoke with the specified color. The task involves 5 keyposes.
17. Stack cups: There are 3 cups. The cup colors are sampled from a set of 20 colors. The agent needs to stack all the other cups on the specified one. The task involves 10 keyposes.
18. Hang cups on the rack: There are 3 mugs and a mug rack. The agent needs to pick up N mugs and place them onto the rack. The task involves 11.5 keyposes.

G.1.4 Real-world tasks

We explain the the real-world tasks and their success conditions in more detail. All tasks take place in a cluttered scene with distractors (random objects that do not participate in the task) which are not mentioned in the descriptions below.

1. Close a box: The end-effector needs to move and hit the lid of an open box so that it closes. The agent is successful if the box closes. The task involves two keyposes.
2. Put a duck in a bowl: There are two toy ducks and two bowls. One of the ducks have to be placed in one of the bowls. The task involves four keyposes.
3. Insert a peg vertically into the hole: The agent needs to detect and grasp a peg, then insert it into a hole that is placed on the ground. The task involves four keyposes.
4. Insert a peg horizontally into the torus: The agent needs to detect and grasp a peg, then insert it into a torus that is placed vertically to the ground. The task involves four keyposes.
5. Put a computer mouse on the pad: There two computer mice and one mousepad.

The agent needs to pick one mouse and place it on the pad. The task involves four keyposes.

6. Open the pen: The agent needs to detect a pen that is attached vertically to the table, grasp its lid and pull it to open the pen. The task involves three keyposes.
7. Press the stapler: The agent needs to reach and press a stapler. The task involves two keyposes.
8. Put grapes in the bowl: The scene contains three vines of grapes of different color and one bowl. The agent needs to pick one vine and place it in the bowl. The task involves four keyposes.
9. Sort the rectangle: Between two rectangle cubes there is space for one more. The task comprises detecting the rectangle to be moved and placing it between the others. It involves four keyposes.
10. Stack blocks with the same shape: The scene contains of several blacks, some of which have rectangular and some cylindrical shape. The task is to pick the same-shape blocks and stack them on top of the first one. It involves eight keyposes.
11. Stack cups: The scene contains three cups of different colors. The agent needs to successfully stack them in any order. The task involves eight keyposes.
12. Put block in a triangle on the plate: The agent needs to detect three blocks of the same color and place them inside a plate to form an equilateral triangle. The task involves 12 keyposes.

The above tasks examine different generalization capabilities of 3D Diffuser Actor, for example multimodality in the solution space (5, 8), order of execution (10, 11, 12), precision (3, 4, 6) and high noise/variance in keyposes (1).

G.1.5 Additional details on baselines

We provide more details of the baselines used in this chapter. For RLBench, we consider the following baselines:

1. C2F-ARM-BC [140], a 3D policy that iteratively voxelizes RGB-D images and predicts actions in a coarse-to-fine manner. Q-values are estimated within each

voxel and the translation action is determined by the centroid of the voxel with the maximal Q-values.

2. PerAct [312], a 3D policy that voxelizes the workspace and detects the next voxel action through global self-attention.
3. Hiveformer [103], a 3D policy that enables attention between features of different history time steps.
4. PolarNet [29], a 3D policy that computes dense point representations for the robot workspace using a PointNext backbone [272].
5. RVT [95], a 3D policy that deploys a multi-view transformer to predict actions and fuses those across views by back-projecting to 3D.
6. Act3D [91], a 3D policy that featurizes the robot's 3D workspace using coarse-to-fine sampling and featurization. We observed that Act3D does not follow the same setup as PerAct on RLBench. Specifically, Act3D uses different 1) 3D object models, 2) success conditions, 3) training/test episodes and 4) maximum numbers of keyposes during evaluation. For fair comparison, we retrain and test Act3D on the same setup.
7. GNFactor [391], a 3D policy that co-optimizes a neural field for reconstructing the 3D voxels of the input scene and a PerAct module for predicting actions based on voxel representations.

We consider the following baselines for CALVIN:

1. MCIL [226], a multi-modal goal-conditioned 2D policy that maps three types of goals–goal images, language instructions and task labels–to a shared latent feature space, and conditions on such latent goals to predict actions.
2. HULC [235], a 2D policy that uses a variational autoencoder to sample a latent plan based on the current observation and task description, then conditions on this latent to predict actions.
3. RT-1 [19], a 2D transformer-based policy that encodes the image and language into a sequence of tokens and employs a Transformer-based architecture that contextualizes these tokens and predicts the arm movement or terminates the episode.
4. RoboFlamingo [184], a 2D policy that adapts existing vision-language models,

which are pre-trained for solving vision and language tasks, to robot control. It uses frozen vision and language foundational models and learns a cross-attention between language and visual features, as well as a recurrent policy that predicts the low-level actions conditioned on the language latents.

5. SuSIE [14], a 2D policy that deploys an large-scale pre-trained image generative model [21] to synthesize visual subgoals based on the current observation and language instruction. Actions are then predicted by a low-level goal-conditioned 2D diffusion policy that models inverse dynamics between the current observation and the predicted subgoal image.
6. GR-1 [359], a 2D policy that first pre-trains an autoregressive Transformer on next frame prediction, using a large-scale video corpus without action annotations. Each video frame is encoded into an 1d vector by average-pooling its visual features. Then, the same architecture is fine-tuned in-domain to predict both actions and future observations.
7. ChainedDiffuser [364], a combination of Act3D [91] that predicts end-effector poses at key frames and a 3D trajectory predictor that generates intermediate trajectories to reach target end-effector poses.
8. 3D Diffusion Policy [393], that encodes sparsely sampled point cloud into 3D representations using a PointNeXt [272] encoder. The 3D representations are average pooled into a 1D feature vector. Then, a UNet conditions on the holistic 3D scene representations and predicts the robot end-effector poses. Notably, in its original implementation, 3D Diffusion Policy is only applied to single-task experimental setup and does not condition on language instructions. For fair comparison, we update the architecture to enable language conditioning. See Section G.1.5 for more details.

Re-training of 3D Diffusion Policy on CALVIN

We compare 3D Diffuser Actor against 3D Diffusion Policy on CALVIN. However, in its original implementation, the 3D Diffusion Policy does not condition on language instructions and is applied only in a single-task setup. For a fair comparison, we enabled language conditioning in the 3D Diffusion Policy by updating its architecture with additional point-cloud-to-language cross-attention layers. We used a language encoder similar to 3D Diffuser Actor to encode language instructions into latent embeddings and applied three cross-attention layers among average pooled point-cloud features and all language tokens. To ensure the maximal performance, we do not use the simplified backbone of DP3 (*Simple DP3*), but the original DP3 architecture.

We adopt the common setup in CALVIN, using both the *front* and *wrist* camera view. Based on the suggestion of the paper [393], we crop a 160×160 and 68×68 bounding box from the depth map of the *front* and *wrist* camera, sampling 1024 points within each bounding box. We train 3D Diffusion Policy with a batch size of 5400 and a total epoch of 3000. Otherwise, we use the default hyper-parameters of 3D Diffusion Policy. We set the horizon of action prediction to 4, the number of executed actions to 3, and the number observed timesteps to 2. During inference, we allow the model to predict 360 times, resulting in a maximum temporal horizon of 1080 actions.

G.1.6 Keypose discovery

For RLBench we use the heuristics from [138, 140]: a pose is a keypose if (1) the end-effector state changes (grasp or release) or (2) the velocity's magnitude approaches zero (often at pre-grasp poses or a new phase of a task). For our real-world experiments we maintain the above heuristics and record pre-grasp poses as well as the poses at the beginning of each phase of a task, e.g., when the end-effector is right above an object of interest. We report the number of keyposes per real-world task in this Appendix (Section G.1.4). Lastly, for CALVIN we adapt the above heuristics to devise a more robust algorithm to discover keyposes. Specifically, we track end-effector state changes and significant changes of motion, i.e. both velocity and acceleration. The Python code is included in our publicly available codebase.

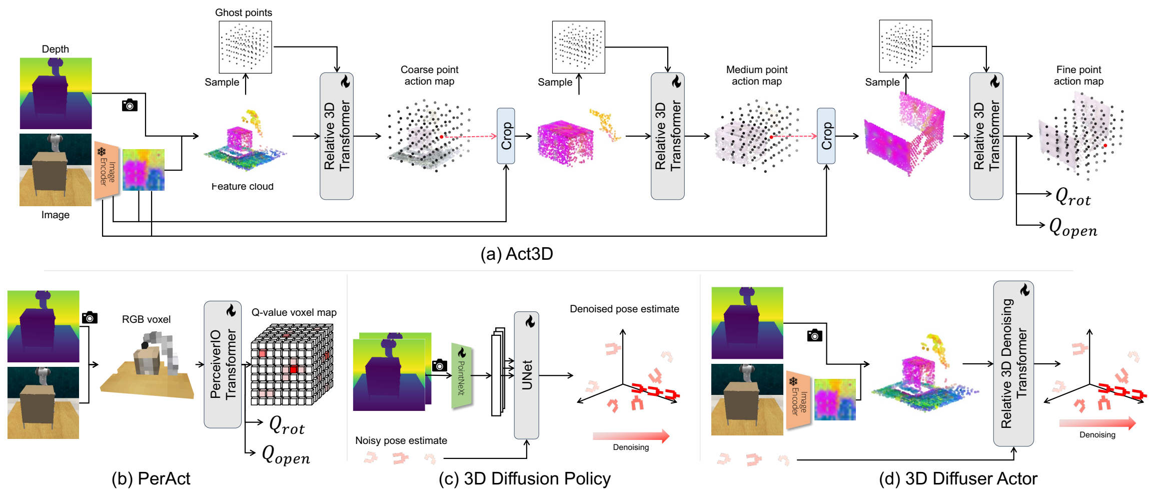


Figure G.2: **Comparison of 3D Diffuser Actor and other methods.** **(a)** Act3D [91] encodes input images into with a 2D feature extractor, lifts 2D feature maps to 3D, samples ghost points within the scene, featurizes ghost points with relative cross attention to the 3D feature cloud, and classifies each ghost point. The position is determined by the XYZ location of the ghost point with the highest score. Act3D then crops the scene with predicted end-effector position, and iterates the same classification procedure. The rotation and openess are predicted with learnable query embeddings. **(b)** PerAct [312] voxelizes input images and uses a Perceiver Transformer [133] to predict the end-effector pose. The position is determined by the XYZ location of the voxel with the highest Q-value. The rotation and openess are predicted from the max-pooled features with MLPs. **(c)** 3D Diffusion Policy [393] encodes 3D point cloud into 1D feature vectors, followed by a A UNet that conditions on point-cloud features and denoises end-effector poses. **(d)** 3D Diffuser Actor uses a similar 3D scene encoder as Act3D. A relative 3D transformer conditions on 3D feature cloud to denoise end-effector poses.

G.2 Additional Method Details

G.2.1 Architectural differences between our model and baselines

We describe the architectural differences between 3D Diffuser Actor and other 3D policies in Figure G.2. We compare 3D Diffuser Actor with Act3D [91], PerAct [312], and 3D Diffusion Policy [393]. PerAct encodes RGB-D images into voxel representations, while 3D Diffusion Policy average pools point cloud representations into a holistic 1D feature vector. Act3D encodes input images with a 2D feature extractor and lifts 2D feature maps to 3D. Act3D determines end-effector poses by iterating a coarse-to-fine classification procedure to identify the XYZ location of the ghost point with the highest classification score. Although our model adopts a similar 3D scene encoder as Act3D, 3D Diffuser Actor uses a relative 3D transformer to denoise end-effector poses.

G.2.2 The formulation of relative attention

To ensure translation invariance in the denoising transformer, we use the rotary positional embeddings [325] to encode relative positional information in attention layers. The attention between query $\mathbf{q}_i$, key $\mathbf{k}_j$ and value $\mathbf{v}_j$ is written as:

$$\text{Attention}(\mathbf{q}_i, \mathbf{k}_j, \mathbf{v}_j) = \frac{\exp e_{i,j}}{\sum_l \exp e_{i,l}} \mathbf{v}_j \tag{G.1}$$

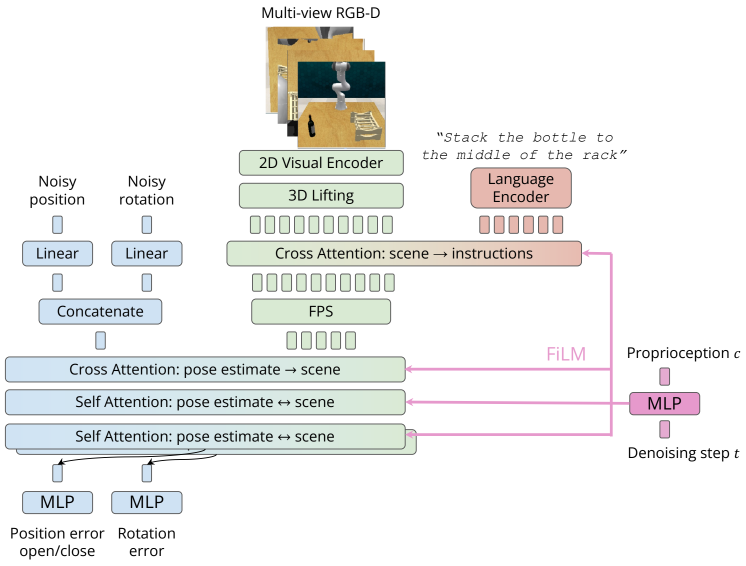
where $e_{i,j} = \mathbf{q}_i^T \mathbf{M}(\mathbf{p}_j - \mathbf{p}_i)\mathbf{k}_j$, $\mathbf{p}_i$ / $\mathbf{p}_j$ denote the positions of the query / key, and $\mathbf{M}$ is a matrix function which depends only on the relative positions of points $\mathbf{p}_i$ and $\mathbf{p}_j$.

G.2.3 Detailed Model Diagram of 3D Diffuser Actor

We present a more detailed architecture diagram of our 3D Diffuser Actor in Figure G.3a. We also show a variant of 3D Diffuser Actor with enhanced language conditioning in Figure G.3b, which achieves SOTA results on CALVIN.

The inputs to our network are i) a stream of RGB-D views; ii) a language instruction; iii) proprioception in the form of current end-effector's poses; iv) the current noisy estimates of position and rotation; v) the diffusion step i. The images are encoded into visual tokens using a pretrained 2D backbone. The depth values are used to "lift" the multi-view tokens into a 3D feature cloud. The language is encoded into feature tokens using a language backbone. The proprioception is represented as learnable tokens with known 3D locations in the scene. The noisy estimates are fed to linear layers that map them to high-dimensional vectors. The denoising step is fed to an MLP.

The visual tokens cross-attend to the language tokens and get residually updated.


Multi-view RGB-D
"Stack the bottle to the middle of the rack"
2D Visual Encoder
3D Lifting
Language Encoder
Noisy position
Noisy rotation
Linear
Linear
Concatenate
Cross Attention: scene → instructions
FPS
FiLM
Proprioception c
MLP
Denoising step t
Cross Attention: pose estimate → scene
Self Attention: pose estimate ↔ scene
Self Attention: pose estimate ↔ scene
MLP
MLP
Position error open/close
Rotation error

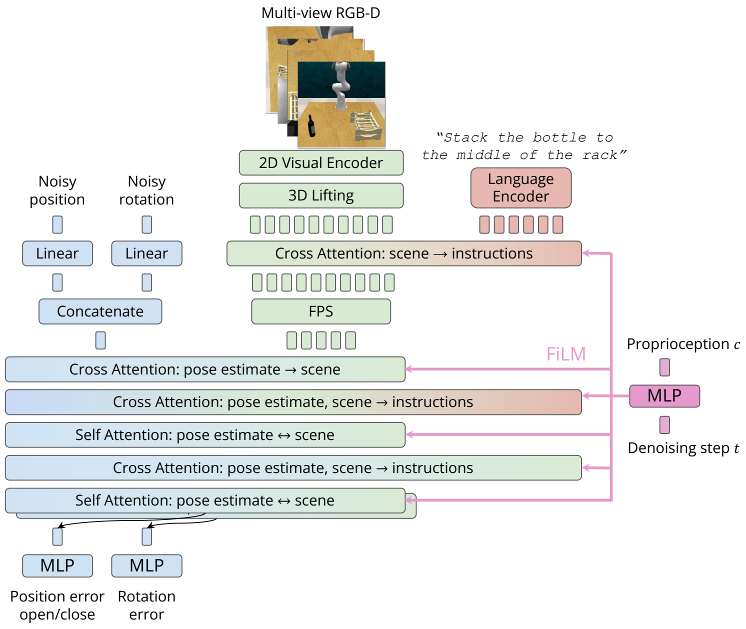

Multi-view RGB-D
"Stack the bottle to the middle of the rack"
2D Visual Encoder
3D Lifting
Language Encoder
Noisy position
Noisy rotation
Linear
Linear
Concatenate
Cross Attention: scene → instructions
FPS
FiLM
Proprioception c
MLP
Denoising step t
Cross Attention: pose estimate → scene
Cross Attention: pose estimate, scene → instructions
Self Attention: pose estimate ↔ scene
Cross Attention: pose estimate, scene → instructions
Self Attention: pose estimate ↔ scene
MLP
MLP
Position error open/close
Rotation error

Figure G.3: **3D Diffuser Actor architecture** in more detail. **Top**: Standard version: the encoded inputs are fed to attention layers that predict the position and rotation error for each trajectory timestep. The language information is fused to the visual stream by allowing the encoded visual feature tokens to attend to language feature tokens. There are two different attention and output heads for position and rotation error respectively. **Bottom**: version with enhanced language conditioning.

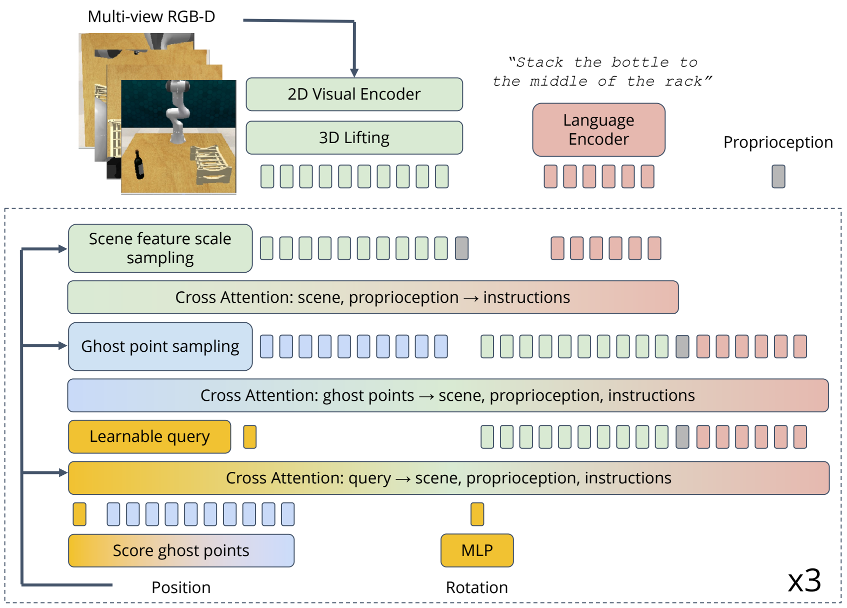

Multi-view RGB-D
2D Visual Encoder
3D Lifting
"Stack the bottle to the middle of the rack"
Language Encoder
Proprioception
Scene feature scale sampling
Cross Attention: scene, proprioception → instructions
Ghost point sampling
Cross Attention: ghost points → scene, proprioception, instructions
Learnable query
Cross Attention: query → scene, proprioception, instructions
Score ghost points
MLP
Position
Rotation
x3

Figure G.4: **Act3D architecture in more detail.** Ghost points and encoded scene feature inputs are sampled based on the current end-effector position. The 3D location of ghost points are even sampled from a fixed-radius sphere, which is centered at the current end-effector position.

The proprioception tokens attend to the visual tokens to contextualize with the scene information. We subsample a number of visual tokens using Farthest Point Sampling (FPS) in order to decrease the computational requirements. The sampled visual tokens, proprioception tokens and noisy position/rotation tokens attend to each other. We modulate the attention using adaptive layer normalization and FiLM [263]. Lastly, the contextualized noisy estimates are fed to MLP to predict the error terms as well as the end-effector's state (open/close).

G.2.4 Detailed Model Diagram of Act3D

To facilitate comparisons in technical detail, we present a detailed architecture diagram of Act3D in Figure G.4. Act3D first encodes posed RGB-D images into 3D scene tokens and samples ghost points uniformly from a fixed-radius sphere, centered at the current end-effector position. Act3D initializes a learnable query and the ghost points with learnable latent embeddings, which are contexturalized by 3D scene tokens using

	Multi-view RLBench	Single-view RLBench	CALVIN
Model			
image_size	256	256	200
embedding_dim	120	120	192
camera_views	4	1	2
FPS : % of sampled tokens	20%	20%	33%
diffusion_timestep	100	100	25
noise_scheduler : position	scaled_linear	scaled_linear	scaled_linear
noise_scheduler : rotation	squaredcos	squaredcos	squaredcos
action_space	absolute pose	absolute pose	relative displacement
Training			
batch_size	240	240	5400
learning_rate	$1e^{-4}$	$1e^{-4}$	$3e^{-4}$
weight_decay	$5e^{-4}$	$5e^{-4}$	$5e^{-3}$
total_epochs	$1.6e^{4}$	$8e^{5}$	90
optimizer	Adam	Adam	Adam
loss weight : w_1	30	30	30
loss weight : w_2	10	10	10
Evaluation			
maximal # of keyposes	25	25	60

Table G.2: **Hyper-parameters of our experiments.** We list the hyper-parameters used for training/evaluating our model on RLBench and CALVIN simulated benchmarks. On RLBench we conduct experiments under multi-view and single-view setup.

3D relative attention (Eqn. G.1). Act3D measures the feature similarity of each ghost point and the learnable query to classify if a ghost point is close to the position of target end-effector pose, which is determined by the XYZ location of the ghost point with the highest score. Act3D then crops the scene with estimated end-effector position, and iterates the same classification procedure. The rotation and openess are predicted with learnable query embeddings.

G.2.5 Hyper-parameters for experiments

Table G.2 summarizes the hyper-parameters used for training/evaluating our model. On CALVIN we observed that our model overfits the training data, resulting in lower test performance. We use higher weight_decay and shorter total_epoch on CALVIN compared to RLBench.

G.2.6 Denoising Diffusion Probabilistic Models

For self-completeness, we provide a formulation of denoising diffusion process. Please check [118] for more details. A diffusion model learns to model a probability distribution $p(x)$ by inverting a process that gradually adds noise to a sample x. For us, x represents a sequence of 3D translations and 3D rotations for the robot's end-effector. The diffusion process is associated with a variance schedule $\{\beta^i \in (0,1)\}_{i=1}^{N}$, which defines how much noise is added at each time step. The noisy version of sample x at time i can then be written as $x^i = \sqrt{\bar{\alpha}^i}x + \sqrt{1-\bar{\alpha}^i}\epsilon$ where $\epsilon \sim \mathcal{N}(\mathbf{0},\mathbf{1})$, is a sample from a Gaussian distribution (with the same dimensionality as x), $\alpha^i = 1-\beta^i$, and $\bar{\alpha}^i = \prod_{j=1}^{i}\alpha^j$. The denoising process is modeled by a neural network $\hat{\epsilon} = \epsilon_\theta(x^i; i)$ that takes as input the noisy sample x^i and the noise level i and tries to predict the noise component ϵ.

Diffusion models can be easily extended to draw samples from a distribution $p(x|\mathbf{c})$ conditioned on input $\mathbf{c}$, which is added as input to the network ϵ_θ. For us, $\mathbf{c}$ is the visual scene captured by one or more calibrated RGB-D images, a language instruction, as well as the proprioceptive information. Given a collection of $\mathcal{D} = \{(x^i, \mathbf{c}^i)\}_{i=1}^{N}$ of end-effector trajectories x^i paired with observation and robot proprioceptive context $\mathbf{c}^i$, the denoising objective becomes:

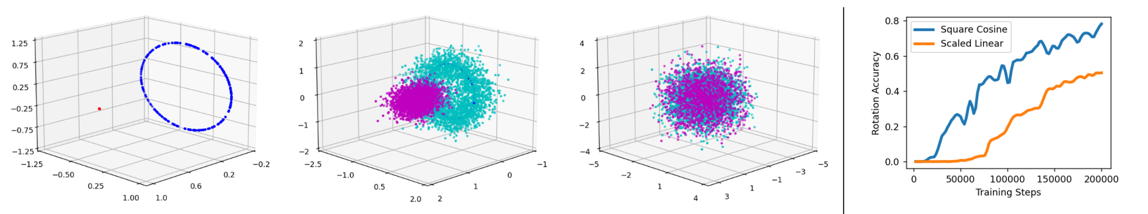
$$\mathcal{L}_{\text{diff}}(\theta;\mathcal{D}) = \tfrac{1}{|\mathcal{D}|}\sum_{x^i,\mathbf{c}^i\in\mathcal{D}} ||\epsilon_\theta(\sqrt{\bar{\alpha}^i}x^i + \sqrt{1-\bar{\alpha}^i}\epsilon, \mathbf{c}^i, i) - \epsilon||. \tag{G.2}$$

This loss corresponds to a reweighted form of the variational lower bound for $\log p(x|\mathbf{c})$ [117].

In order to draw a sample from the learned distribution $p_\theta(x|\mathbf{c})$, we start by drawing a sample $x^N \sim \mathcal{N}(\mathbf{0},\mathbf{1})$. Then, we progressively denoise the sample by iterated application of ϵ_θ N times according to a specified sampling schedule [117, 321], which terminates with x^0 sampled from $p_\theta(x)$:

$$x^{i-1} = \frac{1}{\sqrt{\alpha^i}}\left(x^i - \frac{\beta^i}{\sqrt{1-\bar{\alpha}^i}}\epsilon_\theta(x^i, i, \mathbf{c})\right) + \frac{1-\bar{\alpha}^{i+1}}{1-\bar{\alpha}^i}\beta^i\mathbf{z}, \tag{G.3}$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0},\mathbf{1})$.



(a) Clean (b) Scaled Linear (c) Square Cosine (d) Rotation Accuracy

Figure G.5: **Visualization of noised rotation based on different schedulers.** We split the 6 DoF rotation representations into 2 three-dimension unit-length vectors, and plot the first/second vector as a point in 3D. The noised counterparts are colorized in magenta/cyan. We visualize the rotation of all keyposes in RLBench *insert_peg* task. From left to right, we visualize the (a) clean rotation, (b) noisy rotation with a scaled-linear scheduler, and (c) that with a square cosine scheduler. Lastly, we compare (d) the denoising performance curve of two noise schedulers. Here, accuracy is defined as the percentage of times the absolute rotation error is lower than a threshold of 0.025. Using the square cosine scheduler helps our model to denoise from the pure noise better.

G.2.7 The importance of noise scheduler

Existing diffusion policies [38, 364] often deploy the same noise scheduler for estimate of position and rotation. However, we empirically found that selecting a good noise scheduler is critical. We visualize the clean/noised 6D rotation representations as two three-dimensional unit-length vectors in Figure G.5. We plot each vector as a point in the 3D space. We show that noised rotation vectors generated by the squared linear scheduler cover the space more completely than those by the scaled linear scheduler, resulting in better performance than a square cosine scheduler

Appendix H

3D Flow Actor: An Efficient Flow Matching 3D Policy for Bimanual and Unimanual Control

H.1 Limitations

Despite the tremendous improvements in terms of training and inference cost, 3D generative policies still have limitations.

- 3D policies (including 3DFA) require accurate depth sensors and camera calibration. This is often challenging and thus not available in large-scale real-world imitation learning datasets. We are working on removing the depth and calibration assumption from wrist cameras, that are by construction the most difficult to calibrate.
- As a result of the above, we cannot train large-scale 3D policies on most of the available robotic datasets. This can change in the future, as computer vision makes progress towards metric depth estimation and auto-calibration, or by using architectures that can absorb both 2D and 3D observations [136].
- The policy is still trained with imitation learning, thus it cannot easily discover a new behavior on its own. Combining such 3D policies into a reinforcement learning framework, given their data efficiency, could dramatically speed up

exploration and data collection.

- 3DFA struggles with tasks that require force control or very high precision, such as the hard tasks in our real-world benchmark. The solution to this could be to fine-tune an imitation-trained policy with reinforcement learning.

H.2 Implementation details

Architecture: We closely follow 3DDA [164] in terms of architecture design. We use identical weights to 3DDA, with only the appropriate changes so that we handle and predict bimanual actions of 16 dimensions rather than 8. Specifically, this affects all layers that either project a noisy action token to a high-dimensional latent vector or predict an action token from a latent vector.

For self-completeness, we briefly describe the architecture here: 3DFA uses CLIP [276] to encode each image separately into a feature map, then assigns 3D positions to each 2D feature token using depth and the known camera parameters. The feature maps are merged into a single feature cloud, which is subsampled using density-biased sampling (DBS) to keep 1/4th of the tokens. The distance metric for DBS is computed in the feature space, not the 3D space. We call the subsampled feature cloud "visual tokens".

The noisy trajectory estimate is encoded with a linear layer. Trajectory and visual tokens are concatenated in the sequence dimension and fed to a sequence of 6 self-attention layers. All tokens in this attention use 3D rotary positional embeddings, as used in [91, 164]. Additionally, these layers are modulated by a signal computed by projecting i) the current end-effector pose and ii) the denoising step, into high-dimensional features. This signal is used to compute adaptive normalization parameters.

Lastly, MLPs are used on the output tokens to predict the translation velocity field, rotation velocity field and end-effector opening. Our model is language-conditioned, encodes the instruction with CLIP and then lets the visual and trajectory tokens attend to the language features, as in 3DDA.

Normalization of the output space: 3D relative attention requires that the positional embeddings of all action tokens and visual tokens be expressed in a common coordinate frame. The original 3DDA normalizes the 3D space using the target end-effector's statistics, which does not generalize to bimanual setups where each arm has distinct spatial distributions. Naively aggregating statistics across both arms and normalizing based on that would significantly compress the spatial distribution and harm performance. Instead, we compute positional embeddings in the global unnormalized frame, while still predicting trajectories in the per-single-arm normalized space. This hybrid approach preserves 3DDA's attention mechanism while accommodating the bimanual setup.

Flow hyperparameters: When training a Flow Matching model, we sample a denoising i from a continuous internal between 0 and 1, where $i = 0$ denotes pure noise and $i = 1$ indicates a perfectly clear signal. One hyperparameter is how we sample i during training. We experimented with uniform sampling, beta sampling (as in $\pi0$) and logit-normal sampling (as in [287]). We found beta sampling to consistently underperform the other two options, but little difference between uniform and logit-normal. We adopt logit-normal and sample i as follows: we first sample x from a Gaussian with 0 mean and 1.5 std, then $i = \sigma(x)$, where σ is the sigmoid function.

Data loading details: We replace 3DDA's data loaders by offering the following improvements:

- We change the episodic loading to random keypose sampling. In more detail, the 3DDA codebase loads entire episodes, chunks them and concatenates chunks form different episodes to form a batch. We, on the other hand, sample keyposes across random episodes. To efficiently do this, we used the Python library zarr, to lazily load and access data indices across all episodes simultaneously. This offers the following advantages: i) we avoid loading whole episodes at once to only use a chunk, as this wastes time for data that is not used; ii) we ensure higher diversity in every batch; iii) we ensure a fixed batch size, contrary to 3DDA that concatenates chunks of possibly different sizes.
- We handle data types to ensure faster collating. Specifically, we make sure to always load uint8 images and half-precision depth maps. This significantly speeds up batch formation, especially when large batch sizes are used. The data

is converted to the desired type (float32) after being loaded to the GPU, where data-type conversions are much faster. In contrast, 3DDA loads and batches float32 tensors. We found that handling data types cuts down the loading time by half.

- We move depth unprojection to point cloud and augmentations to GPU. This offers two advantages: i) loading single-channel depth is much faster than three-channel point clouds; ii) it allows for faster, batched operations that are optimized on GPUs.

CUDA graph compilation: This is technique to speed up training, offered by modern versions of PyTorch. The model is compiled as a graph of non-dynamic operations, allowing for PyTorch to optimize all operations under the hood. Making 3DFA compilable required refactoring changes over 3DDA, such as removal of logic branches, CPU operations and custom kernel operations such as FPS. We were able to compile the training mode of 3DFA, by moving language tokenization outside the graph and replacing FPS with a pure PyTorch implementation of DBS, which is both faster and allows for compilation. We do recognize that graph compilation is orthogonal to our efforts to improve the training and inference speed of 3D generative policies, thus we report training times with and without compilation in the main paper, to allow for fair comparisons with future work.

Mixed-precision training: We allow for autocasting operations to half precision when possible. This reduces the memory footprint of our model and allows for large batch sizes without sacrificing performance.

Efficient attention implementation: We use a modern PyTorch implementation of attention that runs optimized C++ code under the hood. This maintains the compatibility of our model and graph compilation. At the same time, this attention implementation is faster by an order of magnitude, especially when combined with large batch sizes and half precision.

Other hyperparameters: We follow the training hyperparameters of 3DDA [164], in terms of optimizer selection, learning rate, etc. We use a constant batch size of 256 keyposes (64 per GPU) for PerAct2 and PerAct, 16 trajectories (one GPU) for the HiveFormer tasks, randomly sampled from different tasks and episodes. We found that our model converges in 300000 training steps on PerAct/PerAct2, contrary to 3DDA that needs 600000 steps. We hypothesize that this is due to the optimized batching scheme we propose. For the HiveFormer, we train single-task models until convergence, which usually takes less than 100000 steps or approximately 4 hours on one GPU.

Keypose discovery We use keyposes only on RLBench, while for our real-world experiments we predict trajectories directly. For our single-arm RLBench experiments, we use the heuristics from [138], while for PerAct2 we use the ones from [100], that adapts those of [138] for the bimanual setup. Specifically we examine each arm separately and a pose is a keypose if (1) the end-effector state changes (grasp or release) or (2) the velocity's magnitude approaches zero (often at pre-grasp poses or a new phase of a task). Once we compute a set keyposes for each arm, we then take the union of the two sets as the bimanual keyposes.

H.3 Additional Experimental Results and Details

H.3.1 Peract2 tasks

We provide an explanation of all 13 PerAct2 tasks we use for self-completeness. All tasks vary the object pose, appearance and semantics. For more details, we refer to the PerAct2 paper [100].

1. Push box: The scene is equipped with a heavy box and a target area. The robot needs to push the box using both arms to move it to the designated area. This task cannot be solved with one robot due to the weight of the box.
2. Lift ball: The scene is equipped with a large ball. The robot needs to grasp the ball using both arms and lift it to a height above 0.95 m. This task is impossible to solve with one robot due to the size of the object.
3. Push buttons: The scene contains three buttons of different colors. The robot needs to push two of the three buttons simultaneously, as specified in the

language instruction.

4. Pick up plate: The robot needs to pick and securely lift a plate. The coordination required to lift the plate without tilting or dropping it.
5. Put item in drawer: The scene contains a furniture with drawers and a small item. The robot needs to open a specific drawer and place an item in it. The correct drawer must be identified as specified in the language instruction.
6. Put bottle in fridge: The robot needs to open a fridge and put a bottle inside.
7. Handover an item: There are multiple items of different colors on the table. The robot needs to pick one with one arm and hand it securely to the arm. The correct item is specified by the language instruction.
8. Pick up laptop: The robot needs to pick up a laptop that is placed on top of a block. This requires first to move the laptop into a position where it can be grasped and then lift it.
9. Straighten rope: The robot needs to straighten a rope so that both ends are placed in distinct target areas.
10. Sweep dustpan: The scene is equipped with a broom, a dust pan, supporting objects and dust. The robot needs to sweep the dust into the dust pan using the broom. The task is considered successfully completed when all the dust is inside the dust pan.
11. Lift tray: The robot needs to lift a tray for more than 1.2m. The tray is originally placed on a holder. An item is on top of the tray and must be balanced while the tray is lifted. The primary challenge lies in coordinating the lifting motion with both arms to maintain the balance of the item on the tray.
12. Handover item (easy): This is a variant of the handover item task (described above), but the same object is always picked.
13. Take tray out of oven: The robot needs to remove a tray that is located inside an oven. This involves opening the oven door and then grasping the tray.

H.3.2 Real-world tasks

We explain the real-world tasks and their success conditions in more detail.

1. Lift ball: The robot needs to use both arms to stabilize and lift a small ball without grasping it.
2. Straighten rope: The robot grasps both ends of the rope and pulls until the rope becomes straight.
3. Pick up plate: The robot manipulates a small plate. The left arm needs to tilt the plate, while the right arm grasps it and lifts it up.
4. Stack bowls: The robot needs to stack two bowls on top of each other.
5. Put marker in cup: The robot needs to use the left arm to reach and fetch the cup, then use the right arm to grasp the marker and put it into the cup.
6. Handover block: The robot uses its left arm to pick up a block and give it to the right arm.
7. Stack blocks: The robot needs to stack two cubes (3cm on each side) on top of each other, which is harder than stacking bowls, since the blocks are smaller.
8. Open marker: The robot needs to grasp the marker with its left arm and then use the right arm to grasp the cap and remove it from the marker.
9. Close ziploc: The robot first grasps one end of the ziploc bag and then grasps the gray slider on the ziploc to close the bag.
10. Insert battery: The robot moves its left arm near the slot for holding and then the right arm grasps the battery and inserts the battery into the slot.

The tasks “stack bowls”, “put marker in cup” and “stack blocks” are bimanual due to the reachability of the objects: the objects are initially placed in locations where only one of the two arms can reach them.

H.3.3 PerAct tasks

PerAct offers a suite of 18 tasks and 249 variations, making it one of the most challenging RLBench-based benchmarks. The variations include instance references, e.g. “open the top/middle/bottom drawer”, color, e.g. “push the blue/purple/etc button”, counting, e.g. “stack 2/3 cups”, and others such as object category, size and shape. We enumerate the 18 tasks here and please refer to the PerAct paper [312] for more details:

1. Open a drawer
2. Slide a block to a colored zone
3. Sweep the dust into a dustpan
4. Take the meat off the grill frame
5. Turn on the water tap
6. Put a block in the drawer
7. Close a jar
8. Drag a block with the stick
9. Stack blocks
10. Screw a light bulb
11. Put the cash in a safe
12. Place a wine bottle on the rack
13. Put groceries in the cardboard
14. Put a block in the shape sorter
15. Push a button
16. Insert onto a peg
17. Stack cups
18. Hang cups on the rack

H.3.4 74 HiveFormer tasks

HiveFormer offers a suite of 74 tasks, grouped into 9 categories. The tasks come without variations, following [91, 102]. We enumerate the groups and tasks here and refer to [91, 102] for more details:

- Planning (can be decomposed into multiple sub-goals): basketball in hoop, put rubbish in bin, meat off grill, meat on grill, change channel, tv on, tower3, push buttons, stack wine.
- Tools (need interaction with a target object): slide block to target, reach and drag, take frame off hanger, water plants, hang frame on hanger, scoop with

spatula, place hanger on rack, move hanger, sweep to dustpan, take plate off colored dish rack, screw nail.

- Long term: wipe desk, stack blocks, take shoes out of box, slide cabinet open and place cups.
- Rotation-invariant: reach target, push button, lamp on, lamp off, push buttons, pick and lift, take lid off saucepan.
- Motion planner (require continuous interaction with the object/environment): toilet seat down, close laptop lid, open box, open drawer, close drawer, close box, phone on base, toilet seat up, put books on bookshelf.
- Multimodal (multiple solutions are possible): pick up cup, turn tap, lift numbered block, beat the buzz, stack cups.
- Precision (high-precision requirements): take usb out of computer, play jenga, insert onto square peg, take umbrella out of umbrella stand, insert usb in computer, straighten rope, pick and lift small, put knife on chopping board, place shape in shape sorter, take toilet roll off stand, put umbrella in umbrella stand, setup checkers.
- Screw (require (un)screwing an object): turn oven on, change clock, open window, open wine bottle.
- Visual Occlusion (large objects occlude certain views): close microwave, close fridge, close grill, open grill, unplug charger, press switch, take money out safe, open microwave, put money in safe, open door, close door, open fridge, open oven, plug charger in power supply

H.3.5 8 ChainedDiffuser tasks

While most of the 74 HiveFormer tasks can be solved by simply predicting keyposes and employing the RRT motion planner to obtain a trajectory to execute, there are several tasks that require continuous interaction with the objects and the environment. For example, when opening a fridge, the end-effector trajectory should respect the mechanical constraints of the door. ChainedDiffuser [364] identified 10 such challenging tasks. PointFlowMatch [39] further investigates trajectory prediction to solve 8 of those 10 tasks. To directly compare 3DFA's ability to predict dense trajectories against those works, we evaluate on the same 8 tasks that both works consider, namely:

1. Unplug charger
2. Close door
3. Open box
4. Open fridge
5. Take frame off hanger
6. Open oven
7. Put books on bookshelf
8. Take shoes out of box

All these tasks require specific motion that cannot be modeled by an RRT planner that connects keyposes with collision-free linear segments. For example, unplugging a charger requires a smooth motion until the charger is out of the socket. Assuming that the end-effector has grasped the charger, the RRT trajectory is not guaranteed to connect the current end-effector pose to the task termination pose with a segment that allows for smoothly extracting the charger from the socket.

H.3.6 Additional details on baselines

On PerAct2, we compare against: (1) ACT [405], a 2D transformer architecture that is trained as a conditional VAE to predict a sequence of actions; (2) RVT-LF [95, 100], that unprojects 2D views to form a point cloud, renders virtual views and feeds them to a transformer to predict the 3D actions for each arm in sequence; (3) PerAct-LF [100, 312], that voxelizes the 3D space and uses a Perceiver [133] architecture to predict the 3D actions for each arm in sequence; (4) PerAct^2 [100], which shares the same architecture as PerAct-LF but predicts the actions for the two arms jointly; (5) AnyBimanual [220], which combines and adapts two pre-trained single-arm PerAct [312] policies; (6) 3D Diffusion Policy (DP3) [393], which encodes 3D scenes with a point cloud encoder and uses a diffusion UNet [293] to predict the 3D actions; (7) KStar Diffuser [224], a diffusion graph convolutional network that regularizes the end-effector pose prediction with predicted body joint angles; (8) PPI [378], a 3D diffusion policy that regularizes action prediction by tracking points sampled from objects of interest; (9) π_0 [15], a state-of-the-art 2D robot generalist that is pre-trained on 10,000 hours of robot data and capable of performing bimanual manipulation. We adapted π_0 to predict end-effector keyposes and fine-tuned it using three cameras.

On the 74 HiveFormer tasks, we compare against: (1) HiveFormer [102], a 3D policy that predicts actions as offsets from the input point cloud; (2) InstructRL [199], a 2D policy that leverages pre-trained vision and language encoders and regresses 6-DoF actions; (3) Act3D [91], a 3D policy that predicts the next action location by iterating between coarse-to-fine sampling and featurization; (4) ChainedDiffuser [364], a two-stage policy that employs Act3D for keypose prediction and a diffusion-based trajectory optimization model to connect the current pose to the next keypose; (5) PointFlowMatch [39], a flow-based 3D policy that encodes the input point cloud into a single vector using PointNet [271] and (6) DP3 [393], discussed in the previous paragraph.

3DFA is trained to predict a keypose-horizon trajectory: it jointly predicts the next end-effector keypose and the dense trajectory until the next keypose in a non-hierarcical, single-forward-pass fashion. HiveFormer, InstructRL and Act3D are trained to predict end-effector keyposes and then use RRT [176] to plan a trajectory. ChainedDiffuser is a two-stage model that predicts the end-effector keypose and then a keypose-conditioned trajectory. PointFlowMatch and DP3 are trained to predict closed-loop trajectories.

H.3.7 Detailed PerAct results

We show detailed results on all 18 tasks for 3DDA and 3DFA in Table H.1. We observe that the average success rate is not fully informative. Although the results in many tasks align, there are statistically significant differences on tasks such as block stacking, putting groceries into cupboard, screwing bulb, sweeping to dustpan and opening drawer. The variance is larger for the two-camera 3DFA, as a mistake may lead to strong occlusions for all cameras, a scenario that is rarer when more cameras are used. Throughout our experimentation, we noticed different sources of large variance:

- First, we identified a very large variance in performance on those tasks across

	Avg. Success	open drawer	slide block	sweep to dustpan	meat off grill	turn tap	put in drawer	close jar	drag stick	stack blocks
3DDA [164]	**81.3**	$89.6_{\pm 4.1}$	$97.6_{\pm 3.2}$	$84.0_{\pm 4.4}$	$\mathbf{96.8}_{\pm 1.6}$	$\mathbf{99.2}_{\pm 1.6}$	$\mathbf{96.0}_{\pm 3.6}$	$96.0_{\pm 2.5}$	$\mathbf{100.0}_{\pm 0.0}$	$\mathbf{68.3}_{\pm 3.3}$
3DFA (2-cam)	78.4	$95.2_{\pm 3.4}$	$\mathbf{98.4}_{\pm 2.2}$	$93.6_{\pm 1.8}$	$\mathbf{96.8}_{\pm 3.4}$	$\mathbf{99.2}_{\pm 1.8}$	$94.4_{\pm 4.6}$	$92.0_{\pm 2.2}$	$99.2_{\pm 1.8}$	$28.0_{\pm 3.6}$
3DFA (4-cam)	80.3	$\mathbf{97.2}_{\pm 2.5}$	$\mathbf{98.4}_{\pm 2.2}$	$\mathbf{99.2}_{\pm 1.8}$	$\mathbf{96.8}_{\pm 1.8}$	$97.8_{\pm 1.8}$	$95.2_{\pm 3.0}$	$\mathbf{98.4}_{\pm 2.2}$	$98.0_{\pm 2.5}$	$41.2_{\pm 3.9}$
		screw bulb	put in safe	place wine	put in cupboard	sort shape	push buttons	insert peg	stack cups	place cups
3DDA [164]		$\mathbf{82.4}_{\pm 2.0}$	$97.6_{\pm 2.0}$	$93.6_{\pm 4.8}$	$\mathbf{85.6}_{\pm 4.1}$	$44.0_{\pm 4.4}$	$98.4_{\pm 2.0}$	$\mathbf{65.6}_{\pm 4.1}$	$47.2_{\pm 8.5}$	$24.0_{\pm 7.6}$
3DFA (2-cam)		$80.0_{\pm 8.0}$	$\mathbf{98.4}_{\pm 2.2}$	$\mathbf{98.4}_{\pm 2.2}$	$68.8_{\pm 7.7}$	$42.0_{\pm 4.9}$	$\mathbf{99.2}_{\pm 1.8}$	$56.4_{\pm 2.2}$	$46.4_{\pm 2.2}$	$24.0_{\pm 2.8}$
3DFA (4-cam)		$72.4_{\pm 4.4}$	$93.6_{\pm 4.1}$	$95.4_{\pm 2.2}$	$78.6_{\pm 5.4}$	$\mathbf{46.4}_{\pm 4.1}$	$97.8_{\pm 1.8}$	$59.2_{\pm 3.0}$	$\mathbf{56.0}_{\pm 2.5}$	$\mathbf{24.2}_{\pm 5.3}$

Table H.1: **Detailed results on the single-arm PerAct benchmark.** 3DFA performs on par with 3DDA on most tasks, even when trained and tested with only two cameras.

checkpoints. For example, for the same model variant, one checkpoint can achieve 80% success in screwing a bulb, while another can achieve 60%. The pattern we observed is that the variance across checkpoints on individual tasks is large, but the average performance on all tasks does not vary significantly. Our hypothesis is that 3DDA and 3DFA are very low-parameter models: during multi-task training, the network may pick a "mode", where it adapts to specific tasks more and underfits others. This could be an avenue for future investigation.

- Second, we tried to fix a checkpoint and evaluate different seeds, which is how we calculate the variance in Table H.1. Most tasks display relatively low variance, e.g., they may hit or miss one episode, but this is exacerbated by the small amount of test episode that PerAct offers: with 25 test episodes, a hit/miss contributes 4% to a task's performance. Interestingly, we find that this variance is not due to the non-deterministic nature of 3DDA or 3DFA, but mostly due to the interaction with the simulator, including the motion planner.
- To isolate the effect of seed, we run the same checkpoint several times with a fixed seed. We still find that internal states in the RLBench simulator and the planner are not controlled by seeding, thus we still observe variance in many tasks.

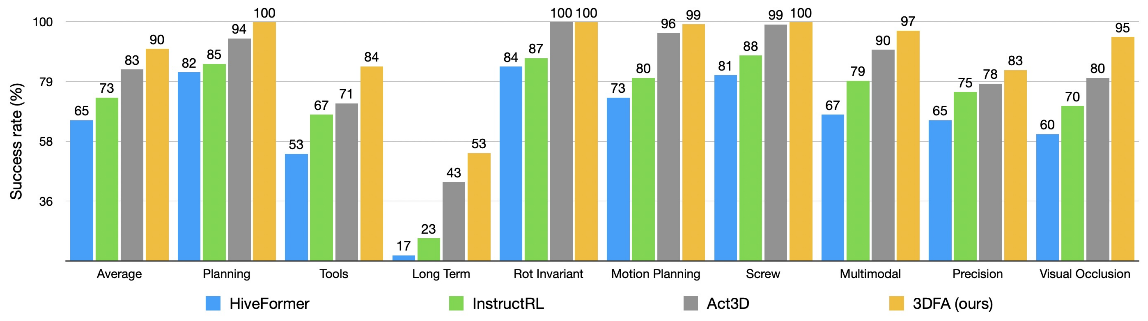


Figure H.1: **Detailed evaluation on 74 RLBench tasks.** grouped into 9 categories. 3DFA achieves a new state of the art on all groups.

H.3.8 Detailed HiveFormer results

We show the results of 3DFA and baselines on the 9 groups in Figure H.1 and the results of 3DFA on all 74 tasks in Table H.2. 3DFA is trained to predict trajectories for all tasks and performs very well on most of them. Predicting trajectories allows 3DFA to not rely on motion planners or hand-designed collision checking that previous works employ [91]. Although 3DFA can solve 57 of 74 tasks with an accuracy higher than 90%, there are still tasks with a much lower success rate. Some of these are due to our choice of cameras. We discuss more details in Section H.3.10.

H.3.9 Detailed real-world results

As shown in Table 9.2, 3DFA excels in performing easy tasks, including *lift ball*, *straighten rope*, *pick up plate*, *stack bowl* and *put marker into bowl*. For more challenging tasks such as *stack blocks*, *open marker*, *close ziploc* and *insert battern*, where objects are smaller/transparent or complex contact dynamics are involved, the performance of our method and baselines decreases, indicating spatial and dynamic reasoning as a clear avenue for future work.

It is worth mentioning that, when 3DFA fails to grasp the related object, we can observe automatic re-attempts on grasping. The model will re-attempt the task until it is successful or until the object moves out of the bounding box.

stack cups	84	beat the buzz	100	lift numbered block	100
turn tap	100	pick up cup	100	wipe desk	37
stack blocks	85	take shoes out of box	71	slide cabinet open and place cup	20
close microwave	100	close fridge	100	close grill	100
unplug charger	99	open grill	100	press switch	100
put money in safe	99	take money out of safe	100	open microwave	100
close door	96	open door	100	open fridge	70
open oven	99	plug charge in power supply	60	slide block to target	100
take frame off hanger	96	reach and drag	100	water plants	79
hang frame on hanger	50	scoop with spatula	86	place hanger on rack	88
move hanger	100	sweep to dustpan	100	take plate off colored dish rack	100
screw nail	26	toilet seat down	100	close laptop lid	100
open drawer	98	open box	100	close drawer	100
phone on base	97	close box	100	reach target	100
toilet seat up	98	push button	100	lamp on	100
put books on bookshelf	99	lamp off	100	pick and lift	100
take lid off saucepan	100	turn oven on	100	change clock	100
open window	100	open wine bottle	100	basketball in hoop	100
put rubbish in bin	100	meat off grill	100	meat on grill	100
change channel	100	tv on	100	tower3	100
push buttons	100	stack wine	100	take usb out of computer	100
insert onto square peg	80	play jenga	100	take umbrella out of umbrella stand	100
insert usb in computer	78	straighten rope	75	pick and lift small	100
put knife on chopping board	99	place shape in shape shorter	40	take toilet roll off stand	97
put umbrella in umbrella stand	25	setup checkers	99	-	-

Table H.2: **Detailed results of 3DFA on all 74 single-arm tasks.**

H.3.10 Failure cases

We analyze the failure modes of 3DFA in both the simulation and the real world. We also discuss the failure modes of our baselines in the real world.

On PerAct2, we notice that most of the errors come from motion planning. For example, in the "handover item" tasks, the RRT-predicted trajectories may not be collision-free when the object is placed near the receiving arm. The task with the lowest performance is "straighten rope". RRT predicts linear segments that do not respect the rope limits. Interestingly, we achieve a much higher success rate for the same task in the real world, where we predict trajectories directly. We did not train a trajectory model on PerAct2 because we found the interaction with the simulator to be particularly slow, making the evaluation of a trajectory prediction model impractical.

To validate the effect of the RRT planner in this setup, we replayed the ground-truth keyposes for the test episodes and use the planner to move between them. We get an average performance of 85.8%, which is only marginally better than our model's. "Pick up plate" and "straighten rope" are still the tasks with most errors, achieving 59% and and 39% respectively.

On PerAct, we observe that the model struggles with understanding the (sometime unseen) variations at test time. For example, it may place the lid on the wrong jar in "close jar" or stack blocks of wrong colors. We verify that on the HiveFormer setup, where there are no variations, 3DFA achieves very high success rates on stacking blocks and cups (Table H.2). This could be mitigated by integrating more powerful representations from a VLM, rather than CLIP. Another source of failure is lower precision than needed in some tasks, such as in "put a block in the shape sorter" and "insert onto peg". 3DFA featurizes all visual observations and subsamples the feature cloud, resulting in a sparser representation. Reducing the number of points is necessary for managing the computational resources. This could be prevented using some VLM guidance to focus around regions of interest for the next action. Lastly, when using two cameras only, occlusions can become an issue.

On the 74 HiveFormer tasks, high-precision requirements and occlusions are the main issues. 3DFA achieves its lowest performance on the "long-term" task group. However, this is due to heavy occlusions caused by using the front and wrist cameras only. Specifically, for "slide cabinet open and place cup", the cup is most of the times hidden from the front camera, as it is occluded by the cabinet. 3DFA always opens the cabinet but then struggles to find the cup. To verify, we trained 3DFA using the overhead camera for this task and achieved 74%. Another example of heavy occlusion is "open fridge". Some examples of high-precision tasks that 3DFA struggles are "hang frame on hanger", "place shape in shape shorter" and "put umbrella in umbrella stand".

In the real world experiments, most failure cases of 3DFA in simple tasks are caused by last-centimeter errors that move the objects towards an undesired direction. Examples include slipping the ball or not raising the other edge of the plate high enough. For hard tasks, the success rate is much lower; we think this is caused by occlusion from the robot or objects in the "handover block", "open marker" and "insert battery" tasks. We also discovered a 2cm error in the robot's forward kinematics, which significantly reduces the performance on the aforementioned tasks that require precise operation. This error is exacerbated when the arm leans forward too much. We show videos of different failure cases for our model on our website.

For baselines, we found that π_0 [15] needs to keep the relevant object always appearing on the wrist camera's view, otherwise it cannot finish the grasping part; this problem is obvious in "handover block", "open marker" and "insert battery" when the wrist camera view is blocked by the object in hand or the robot itself. Most of π_0 failure cases are in discovering the 3D relationship of objects in the task. For example, in the two stacking tasks, the object cannot be placed directly above the other, and in the "insert marker into the cup" task, the model cannot find the correct position to drop the marker. On the other hand, iDP3 [392] has a high success rate in finding the 3D relationship between objects, while most of its failure cases are caused by failed grasps.

Bibliography

[1] Ahmed Abdelreheem, Ujjwal Upadhyay, Ivan Skorokhodov, Rawan Al Yahya, Jun Chen, and Mohamed Elhoseiny. 3DRefTransformer: Fine-Grained Object Identification in Real-World Scenes Using Natural Language. In *Proc. WACV*, 2022.

[2] Panos Achlioptas, Ahmed Abdelreheem, Fei Xia, Mohamed Elhoseiny, and Leonidas Guibas. ReferIt3D: Neural Listeners for Fine-Grained 3D Object Identification in Real-World Scenes. In *Proc. ECCV*, 2020.

[3] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[4] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B. Tenenbaum, T. Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? *ArXiv*, abs/2211.15657, 2022.

[5] Anurag Ajay, Seungwook Han, Yilun Du, Shaung Li, Gupta Abhi, Tommi Jaakkola, Josh Tenenbaum, Leslie Kaelbling, Akash Srivastava, and Pulkit Agrawal. Compositional foundation models for hierarchical planning. *arXiv preprint arXiv:2309.08587*, 2023.

[6] Ahmed Akakzia, Cédric Colas, Pierre-Yves Oudeyer, Mohamed Chetouani, and Olivier Sigaud. Grounding Language to Autonomously-Acquired Skills via Goal Generation. In *ICLR 2021 - Ninth International Conference on Learning Representation*, Vienna / Virtual, Austria, May 2021. URL https://hal.inria.fr/hal-03121146.

[7] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering. In *Proc. CVPR*, 2018.

[8] Iro Armeni, Sasha Sax, Amir R Zamir, and Silvio Savarese. Joint 2d-3d-semantic data for indoor scene understanding. *arXiv preprint arXiv:1702.01105*, 2017.

[9] Ali Athar, Jonathon Luiten, Alexander Hermans, Deva Ramanan, and Bastian

Leibe. Hodor: High-level object descriptors for object re-segmentation in video learned from static images, 2021. URL https://arxiv.org/abs/2112.09131.

[10] Christopher G. Atkeson and Stefan Schaal. Robot learning from demonstration. In *International Conference on Machine Learning*, 1997.

[11] Amir Bar, Yossi Gandelsman, Trevor Darrell, Amir Globerson, and Alexei A Efros. Visual prompting via image inpainting. *NeurIPS*, 2022.

[12] Moshe Bar. The proactive brain: Using analogies and associations to generate predictions. *Trends in Cognitive Sciences*, 11(7):280–289, 2007. doi: 10.1016/j.tics.2007.05.005.

[13] Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

[14] Kevin Black, Mitsuhiko Nakamoto, Pranav Atreya, Homer Walke, Chelsea Finn, Aviral Kumar, and Sergey Levine. Zero-shot robotic manipulation with pretrained image-editing diffusion models. *arXiv preprint arXiv:2310.10639*, 2023.

[15] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π0: A vision-language-action flow model for general robot control, 2024. *arXiv preprint arXiv:2410.24164*, 2024.

[16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. URL http://arxiv.org/abs/1604.07316.

[17] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. *CoRR*, abs/2112.04426, 2021. URL https://arxiv.org/abs/2112.04426.

[18] Max Braun, Noémie Jaquier, Leonel Rozo, and Tamim Asfour. Riemannian flow matching policy for robot motion learning. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5144–5151. IEEE, 2024.

[19] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[20] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023.

[21] Tim Brooks, Aleksander Holynski, and Alexei A Efros. Instructpix2pix: Learning to follow image editing instructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18392–18402, 2023.

[22] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[23] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-End Object Detection with Transformers. In *Proc. ECCV*, 2020.

[24] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *arXiv preprint arXiv:1709.06158*, 2017.

[25] Dave Zhenyu Chen, Angel Chang, and Matthias Nießner. ScanRefer: 3D Object Localization in RGB-D Scans using Natural Language. In *Proc. ECCV*, 2020.

[26] Huayu Chen, Cheng Lu, Chengyang Ying, Hang Su, and Jun Zhu. Offline reinforcement learning via high-fidelity generative behavior modeling, 2023.

[27] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, P. Abbeel, A. Srinivas, and Igor Mordatch. Decision transformer:

Reinforcement learning via sequence modeling. In *Neural Information Processing Systems*, 2021.

[28] Shaoyu Chen, Jiemin Fang, Qian Zhang, Wenyu Liu, and Xinggang Wang. Hierarchical aggregation for 3d instance segmentation. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 15447–15456, 2021.

[29] Shizhe Chen, Ricardo Garcia Pinel, Cordelia Schmid, and Ivan Laptev. Polarnet: 3d point clouds for language-guided robotic manipulation. *ArXiv*, abs/2309.15596, 2023.

[30] Wenhu Chen, Pat Verga, Michiel de Jong, John Wieting, and William Cohen. Augmenting pre-trained language models with qa-memory for open-domain question answering, 2022. URL https://arxiv.org/abs/2204.04581.

[31] Xiaobai Chen, Aleksey Golovinskiy, and Thomas A. Funkhouser. A benchmark for 3d mesh segmentation. *ACM Trans. Graph.*, 28:73, 2009.

[32] Xinpeng Chen, Lin Ma, Jingyuan Chen, Zequn Jie, Wei Liu, and Jiebo Luo. Real-time referring expression comprehension by single-stage grounding network. *ArXiv*, abs/1812.03426, 2018.

[33] Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. UNITER: UNiversal Image-TExt Representation Learning. In *Proc. ECCV*, 2020.

[34] Bowen Cheng, Ishan Misra, Alexander G. Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. *CoRR*, abs/2112.01527, 2021. URL https://arxiv.org/abs/2112.01527.

[35] Bowen Cheng, Alexander G. Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation. In *NeurIPS*, 2021.

[36] Bowen Cheng, Ishan Misra, Alexander G. Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. 2022.

[37] Sonia Chernova and Manuela Veloso. Confidence-based policy learning from demonstration using gaussian mixture models. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9788190426275. doi: 10.1145/1329125.1329407. URL https://doi.org/10.1145/1329125.1329407.

[38] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

[39] Eugenio Chisari, Nick Heppert, Max Argus, Tim Welschehold, Thomas Brox, and Abhinav Valada. Learning robotic manipulation policies from point clouds with conditional flow matching. *arXiv preprint arXiv:2409.07343*, 2024.

[40] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks, 2019.

[41] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3075–3084, 2019.

[42] Cédric Colas, Tristan Karch, Nicolas Lair, Jean-Michel Dussoux, Clément Moulin-Frier, Peter Ford Dominey, and Pierre-Yves Oudeyer. Language as a cognitive tool to imagine goals in curiosity-driven exploration. *CoRR*, abs/2002.09253, 2020. URL https://arxiv.org/abs/2002.09253.

[43] Cédric Colas, Tristan Karch, Clément Moulin-Frier, and Pierre-Yves Oudeyer. Vygotskian autotelic artificial intelligence: Language and culture internalization for human-like ai, 2022. URL https://arxiv.org/abs/2206.01134.

[44] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. Reducing the barrier to entry of complex robotic software: a moveit! case study. *arXiv preprint arXiv:1404.3785*, 2014.

[45] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

[46] Zichen Jeff Cui, Yibin Wang, Nur Muhammad (Mahi) Shafiullah, and Lerrel Pinto. From play to policy: Conditional behavior generation from uncurated robot data. *ArXiv*, abs/2210.10047, 2022.

[47] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5828–5839, 2017.

[48] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes, 2017. URL http://arxiv.org/abs/1702.04405. cite arxiv:1702.04405.

[49] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas A. Funkhouser, and Matthias Nießner. ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes. In *Proc. CVPR*, 2017.

[50] Yilun Dai, Mengjiao Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Josh Tenenbaum, Dale Schuurmans, and Pieter Abbeel. Learning universal policies via text-guided

video generation. *arXiv preprint arXiv:2302.00111*, 2023.

[51] Murtaza Dalal, Ajay Mandlekar, Caelan Reed Garrett, Ankur Handa, Ruslan Salakhutdinov, and Dieter Fox. Imitating task and motion planning with visuomotor transformers. In *Conference on Robot Learning*, 2023.

[52] Rajarshi Das, Ameya Godbole, Ankita Naik, Elliot Tower, Robin Jia, Manzil Zaheer, Hannaneh Hajishirzi, and Andrew McCallum. Knowledge base question answering by case-based reasoning over subgraphs, 2022. URL https://arxiv.org/abs/2202.10610.

[53] Michiel de Jong, Yury Zemlyanskiy, Nicholas FitzGerald, Fei Sha, and William Cohen. Mention memory: incorporating textual knowledge into transformers through entity mention attention. *CoRR*, abs/2110.06176, 2021. URL https://arxiv.org/abs/2110.06176.

[54] Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Procthor: Large-scale embodied ai using procedural generation. *Advances in Neural Information Processing Systems*, 35:5982–5994, 2022.

[55] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, K. Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proc. CVPR*, 2009.

[56] Jiajun Deng, Zhengyuan Yang, Tianlang Chen, Wengang Zhou, and Houqiang Li. Transvg: End-to-end visual grounding with transformers. In *Proc. ICCV*, 2021.

[57] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. NAACL*, 2019.

[58] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. *ArXiv*, abs/2105.05233, 2021.

[59] Haoran Ding, Noémie Jaquier, Jan Peters, and Leonel Rozo. Fast and robust visuomotor riemannian flow matching policy. *arXiv preprint arXiv:2412.10855*, 2024.

[60] Runyu Ding, Jihan Yang, Chuhui Xue, Wenqing Zhang, Song Bai, and Xiaojuan Qi. Pla: Language-driven open-vocabulary 3d scene understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7010–7019, 2023.

[61] Yiming Ding, Carlos Florensa, Pieter Abbeel, and Mariano Phielipp. Goal-conditioned imitation learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.,

2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/c8d3a760ebab631565f8509d84b3b3f1-Paper.pdf.

[62] Bin Dong, Fangao Zeng, Tiancai Wang, Xiangyu Zhang, and Yichen Wei. Solq: Segmenting objects by learning queries. *Advances in Neural Information Processing Systems*, 34:21898–21909, 2021.

[63] Li Dong and Mirella Lapata. Language to logical form with neural attention. *arXiv preprint arXiv:1601.01280*, 2016.

[64] Shichao Dong, Fayao Liu, and Guosheng Lin. Leveraging large-scale pretrained vision foundation models for label-efficient 3d point cloud segmentation. *arXiv preprint arXiv:2311.01989*, 2023.

[65] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[66] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

[67] Yilun Du, Toru Lin, and Igor Mordatch. Model based planning with energy based models. *CoRR*, abs/1909.06878, 2019. URL http://arxiv.org/abs/1909.06878.

[68] Yilun Du, Shuang Li, and Igor Mordatch. Compositional visual generation with energy based models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6637–6647. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/49856ed476ad01fcff881d57e161d73f-Paper.pdf.

[69] Yilun Du, Shuang Li, Joshua Tenenbaum, and Igor Mordatch. Improved contrastive divergence training of energy based models. *arXiv preprint arXiv:2012.01316*, 2020.

[70] Yilun Du, Shuang Li, Joshua B. Tenenbaum, and Igor Mordatch. Improved contrastive divergence training of energy based models. *CoRR*, abs/2012.01316, 2020. URL https://arxiv.org/abs/2012.01316.

[71] Yilun Du, Mengjiao Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Joshua B. Tenenbaum, Dale Schuurmans, and P. Abbeel. Learning universal policies via text-guided video generation. *ArXiv*, abs/2302.00111, 2023.

[72] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and

Xavier Bresson. Graph neural networks with learnable structural and positional representations. *arXiv preprint arXiv:2110.07875*, 2021.

[73] Y. Eldar, M. Lindenbaum, M. Porat, and Y.Y. Zeevi. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing*, 6(9): 1305–1315, 1997. doi: 10.1109/83.623193.

[74] Hao Fang, Saurabh Gupta, Forrest Iandola, Rupesh K Srivastava, Li Deng, Piotr Dollár, Jianfeng Gao, Xiaodong He, Margaret Mitchell, John C Platt, et al. From Captions to Visual Concepts and Back. In *Proc. CVPR*, 2015.

[75] Hao-Shu Fang, Hongjie Fang, Zhenyu Tang, Jirong Liu, Chenxi Wang, Junbo Wang, Haoyi Zhu, and Cewu Lu. Rh20t: A comprehensive robotic dataset for learning diverse skills in one-shot. *arXiv preprint arXiv:2307.00595*, 2023.

[76] Xiaolin Fang, Caelan Reed Garrett, Clemens Eppner, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Dieter Fox. Dimsam: Diffusion models as samplers for task and motion planning under partial observability. *arXiv preprint arXiv:2306.13196*, 2023.

[77] Mingtao Feng, Zhen Li, Qi Li, Liang Zhang, XiangDong Zhang, Guangming Zhu, Hui Zhang, Yaonan Wang, and Ajmal Mian. Free-form Description Guided 3D Visual Graph Network for Object Grounding in Point Cloud. In *Proc. ICCV*, 2021.

[78] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/finn17a.html.

[79] Matthew Fisher, Daniel Ritchie, Manolis Savva, Thomas Funkhouser, and Pat Hanrahan. Example-based synthesis of 3d object arrangements. *ACM Transactions on Graphics (TOG)*, 31(6):1–11, 2012.

[80] Adam Fishman, Adithya Murali, Clemens Eppner, Bryan N. Peele, Byron Boots, and Dieter Fox. Motion policy networks. In *Conference on Robot Learning*, 2022.

[81] Pete Florence, Corey Lynch, Andy Zeng, Oscar Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. *CoRR*, abs/2109.00137, 2021. URL https://arxiv.org/abs/2109.00137.

[82] Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Conference on Robot Learning*, pages 158–168. PMLR, 2022.

[83] Zipeng Fu, Tony Z Zhao, and Chelsea Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. *arXiv preprint arXiv:2401.02117*, 2024.

[84] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation, 2024.

[85] Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. Multimodal Compact Bilinear Pooling for Visual Question Answering and Visual Grounding. In *Proc. EMNLP*, 2016.

[86] Niklas Funk, Julen Urain, Joao Carvalho, Vignesh Prasad, Georgia Chalvatzaki, and Jan Peters. Actionflow: Equivariant, accurate, and efficient policies with spatially symmetric flow matching. *arXiv preprint arXiv:2409.04576*, 2024.

[87] Zhe Gan, Yen-Chun Chen, Linjie Li, Chen Zhu, Yu Cheng, and Jingjing Liu. Large-Scale Adversarial Training for Vision-and-Language Representation Learning. In *Proc. NeurIPS*, 2020.

[88] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Strip-stream: Integrating symbolic planners and blackbox samplers. *CoRR*, abs/1802.08705, 2018. URL http://arxiv.org/abs/1802.08705.

[89] Haoran Geng, Ziming Li, Yiran Geng, Jiayi Chen, Hao Dong, and He Wang. Partmanip: Learning cross-category generalizable part manipulation policy from point cloud observations. *ArXiv*, abs/2303.16958, 2023.

[90] Kyle Genova, Xiaoqi Yin, Abhijit Kundu, Caroline Pantofaru, Forrester Cole, Avneesh Sud, Brian Brewington, Brian Shucker, and Thomas Funkhouser. Learning 3d semantic segmentation with only 2d image supervision. In *2021 International Conference on 3D Vision (3DV)*, pages 361–372. IEEE, 2021.

[91] Theophile Gervet, Zhou Xian, Nikolaos Gkanatsios, and Katerina Fragkiadaki. Act3d: 3d feature field transformers for multi-task robotic manipulation. *CoRL*, 2023.

[92] Rohit Girdhar, Mannat Singh, Nikhila Ravi, Laurens van der Maaten, Armand Joulin, and Ishan Misra. Omnivore: A single model for many visual modalities. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16102–16112, 2022.

[93] Nikolaos Gkanatsios, Ayush Jain, Zhou Xian, Yunchu Zhang, Christopher Atkeson, and Katerina Fragkiadaki. Energy-based models as zero-shot planners for compositional scene rearrangement. *arXiv preprint arXiv:2304.14391*, 2023.

[94] Nikolaos Gkanatsios, Mayank Kumar Singh, Zhaoyuan Fang, Shubham Tulsiani, and Katerina Fragkiadaki. Analogy-forming transformers for few-shot 3d parsing. *ArXiv*, abs/2304.14382, 2023.

[95] Ankit Goyal, Jie Xu, Yijie Guo, Valts Blukis, Yu-Wei Chao, and Dieter Fox. Rvt: Robotic view transformer for 3d object manipulation. *arXiv preprint arXiv:2306.14896*, 2023.

[96] Ben Graham. Sparse 3d convolutional neural networks, 2015.

[97] Jennifer Grannen, Yilin Wu, Suneel Belkhale, and Dorsa Sadigh. Learning bimanual scooping policies for food acquisition. In *Conference on Robot Learning*, 2022.

[98] Jennifer Grannen, Yilin Wu, Brandon Vu, and Dorsa Sadigh. Stabilize to act: Learning to coordinate for bimanual manipulation. *CoRL*, 2023.

[99] Will Grathwohl, Kuan-Chieh Wang, Jörn-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. Your classifier is secretly an energy based model and you should treat it like one. *CoRR*, abs/1912.03263, 2019. URL http://arxiv.org/abs/1912.03263.

[100] Markus Grotz, Mohit Shridhar, Tamim Asfour, and Dieter Fox. Peract2: A perceiver actor framework for bimanual manipulation tasks. *arXiv preprint arXiv:2407.00278*, 2024.

[101] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393*, 2016.

[102] Pierre-Louis Guhur, Shizhe Chen, Ricardo Garcia Pinel, Makarand Tapaswi, Ivan Laptev, and Cordelia Schmid. Instruction-driven history-aware policies for robotic manipulations. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=h0Yb0U_-Tki.

[103] Pierre-Louis Guhur, Shizhe Chen, Ricardo Garcia Pinel, Makarand Tapaswi, Ivan Laptev, and Cordelia Schmid. Instruction-driven history-aware policies for robotic manipulations. In *Conference on Robot Learning*, pages 175–187. PMLR, 2023.

[104] Huy Ha and Shuran Song. Semantic abstraction: Open-world 3d scene understanding from 2d vision-language models. In *6th Annual Conference on Robot Learning*, 2022.

[105] Lei Han, Tian Zheng, Lan Xu, and Lu Fang. Occuseg: Occupancy-aware 3d instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2940–2949, 2020.

[106] Philippe Hansen-Estruch, Ilya Kostrikov, Michael Janner, Jakub Grudzien Kuba, and Sergey Levine. Idql: Implicit q-learning as an actor-critic method with diffusion policies. *arXiv preprint arXiv:2304.10573*, 2023.

[107] Adam W Harley, Shrinidhi K Lakshmikanth, Fangyu Li, Xian Zhou, Hsiao-

Yu Fish Tung, and Katerina Fragkiadaki. Learning from unlabelled videos using contrastive predictive neural 3d mapping. *arXiv preprint arXiv:1906.03764*, 2019.

[108] Karol Hausman, Yevgen Chebotar, Stefan Schaal, Gaurav S. Sukhatme, and Joseph J. Lim. Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets. *CoRR*, abs/1705.10479, 2017. URL http://arxiv.org/abs/1705.10479.

[109] Dailan He, Yusheng Zhao, Junyu Luo, Tianrui Hui, Shaofei Huang, Aixi Zhang, and Si Liu. TransRefer3D: Entity-and-Relation Aware Transformer for Fine-Grained 3D Visual Grounding. In *Proc. ACMMM*, 2021.

[110] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proc. CVPR*, 2016.

[111] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[112] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[113] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[114] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. In *Proc. ICCV*, 2017.

[115] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009, 2022.

[116] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016. URL http://arxiv.org/abs/1606.03476.

[117] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *CoRR*, abs/2006.11239, 2020. URL https://arxiv.org/abs/2006.11239.

[118] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[119] Ronghang Hu, Marcus Rohrbach, Jacob Andreas, Trevor Darrell, and Kate Saenko. Modeling Relationships in Referential Expressions with Compositional

Modular Networks. In *Proc. CVPR*, 2017.

[120] Wenbo Hu, Hengshuang Zhao, Li Jiang, Jiaya Jia, and Tien-Tsin Wong. Bidirectional projection network for cross dimension scene understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14373–14382, 2021.

[121] Yucheng Hu, Yanjiang Guo, Pengchao Wang, Xiaoyu Chen, Yen-Jen Wang, Jianke Zhang, Koushil Sreenath, Chaochao Lu, and Jianyu Chen. Video prediction policy: A generalist robot policy with predictive visual representations, 2025. URL https://arxiv.org/abs/2412.14803.

[122] Haojie Huang, Owen Howell, Xupeng Zhu, Dian Wang, Robin Walters, and Robert Platt. Fourier transporter: Bi-equivariant robotic manipulation in 3d. In *ICLR*, 2024.

[123] Jingwei Huang, Haotian Zhang, Li Yi, Thomas Funkhouser, Matthias Nießner, and Leonidas J Guibas. Texturenet: Consistent local parametrizations for learning from high-resolution signals on meshes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4440–4449, 2019.

[124] Pin-Hao Huang, Han-Hung Lee, Hwann-Tzong Chen, and Tyng-Luh Liu. Text-Guided Graph Neural Networks for Referring 3D Instance Segmentation. In *Proc. AAAI*, 2021.

[125] Siyuan Huang, Zan Wang, Puhao Li, Baoxiong Jia, Tengyu Liu, Yixin Zhu, Wei Liang, and Song-Chun Zhu. Diffusion-based generation, optimization, and planning in 3d scenes. *ArXiv*, abs/2301.06015, 2023.

[126] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022.

[127] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models, 2022. URL https://arxiv.org/abs/2207.05608.

[128] Wenlong Huang, Fei Xia, Dhruv Shah, Danny Driess, Andy Zeng, Yao Lu, Pete Florence, Igor Mordatch, Sergey Levine, Karol Hausman, et al. Grounded decoding: Guiding text generation with grounded models for robot control. *arXiv preprint arXiv:2303.00855*, 2023.

[129] Zhiao Huang, Yuanming Hu, Tao Du, Siyuan Zhou, Hao Su, Joshua B Tenenbaum, and Chuang Gan. Plasticinelab: A soft-body manipulation benchmark

with differentiable physics. *arXiv preprint arXiv:2104.03311*, 2021.

[130] Gautier Izacard and Edouard Grave. Distilling Knowledge from Reader to Retriever for Question Answering. In *ICLR 2021 - 9th International Conference on Learning Representations*, Vienna, Austria, May 2021. URL https://hal.archives-ouvertes.fr/hal-03463398.

[131] Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Few-shot learning with retrieval augmented language models, 2022. URL https://arxiv.org/abs/2208.03299.

[132] Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and João Carreira. Perceiver: General Perception with Iterative Attention. In *Proc. ICML*, 2021.

[133] Andrew Jaegle, Felix Gimeno, Andrew Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver: General perception with iterative attention, 2021.

[134] Ayush Jain, Nikolaos Gkanatsios, Ishita Mediratta, and Katerina Fragkiadaki. Bottom up top down detection transformers for language grounding in images and point clouds. In *European Conference on Computer Vision*, pages 417–433. Springer, 2022.

[135] Ayush Jain, Nikolaos Gkanatsios, Ishita Mediratta, and Katerina Fragkiadaki. Bottom up top down detection transformers for language grounding in images and point clouds. In *European Conference on Computer Vision*, pages 417–433. Springer, 2022.

[136] Ayush Jain, Pushkal Katara, Nikolaos Gkanatsios, Adam W. Harley, Gabriel Sarch, Kriti Aggarwal, Vishrav Chaudhary, and Katerina Fragkiadaki. Odin: A single model for 2d and 3d segmentation, 2024. URL https://arxiv.org/abs/2401.02416.

[137] Ayush Jain, Pushkal Katara, Nikolaos Gkanatsios, Adam W. Harley, Gabriel H. Sarch, Kriti Aggarwal, Vishrav Chaudhary, and Katerina Fragkiadaki. Odin: A single model for 2d and 3d perception. *ArXiv*, abs/2401.02416, 2024.

[138] Stephen James and Andrew J Davison. Q-attention: Enabling efficient learning for vision-based robotic manipulation. *IEEE Robotics and Automation Letters*, 7(2):1612–1619, 2022.

[139] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.

[140] Stephen James, Kentaro Wada, Tristan Laidlow, and Andrew J Davison. Coarse-to-fine q-attention: Efficient learning for visual robotic manipulation via dis-

cretisation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13739–13748, 2022.

[141] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002. PMLR, 2022.

[142] Michael Janner, Qiyang Li, and Sergey Levine. Reinforcement learning as one big sequence modeling problem. In *Neural Information Processing Systems*, 2021.

[143] Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

[144] Maximilian Jaritz, Jiayuan Gu, and Hao Su. Multi-view pointnet for 3d scene understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[145] Daniel Jarrett, Ioana Bica, and Mihaela van der Schaar. Strictly batch imitation learning by energy-based distribution matching. *Advances in Neural Information Processing Systems*, 33:7354–7365, 2020.

[146] Krishna Murthy Jatavallabhula, Alihusein Kuwajerwala, Qiao Gu, Mohd Omama, Tao Chen, Shuang Li, Ganesh Iyer, Soroush Saryazdi, Nikhil Keetha, Ayush Tewari, et al. Conceptfusion: Open-set multimodal 3d mapping. *arXiv preprint arXiv:2302.07241*, 2023.

[147] Chiyu Jiang, Andre Cornman, Cheolho Park, Benjamin Sapp, Yin Zhou, and Dragomir Anguelov. Motiondiffuser: Controllable multi-agent motion prediction using diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9644–9653, June 2023.

[148] Jian-Jian Jiang, Xiao-Ming Wu, Yi-Xiang He, Ling-An Zeng, Yi-Lin Wei, Dandan Zhang, and Wei-Shi Zheng. Rethinking bimanual robotic manipulation: Learning with decoupled interaction framework. *arXiv preprint arXiv:2503.09186*, 2025.

[149] Li Jiang, Hengshuang Zhao, Shaoshuai Shi, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Pointgroup: Dual-set point grouping for 3d instance segmentation. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4866–4875, 2020.

[150] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi (Jim) Fan. Vima: General robot manipulation with multimodal prompts. *ArXiv*, abs/2210.03094, 2022.

[151] Justin Johnson, Andrej Karpathy, and Li Fei-Fei. DenseCap: Fully Convolutional Localization Networks for Dense Captioning. In *Proc. CVPR*, 2016.

[152] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2901–2910, 2017.

[153] Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs. *CoRR*, abs/1804.01622, 2018. URL http://arxiv.org/abs/1804.01622.

[154] R. K. Jones, Aalia Habib, and Daniel Ritchie. Shred: 3d shape region decomposition with learned local operations. *ArXiv*, abs/2206.03480, 2022.

[155] Rishabh Kabra, Chris Burgess, Loic Matthey, Raphael Lopez Kaufman, Klaus Greff, Malcolm Reynolds, and Alexander Lerchner. Multi-object datasets. https://github.com/deepmind/multi-object-datasets/, 2019.

[156] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*, pages 1470–1477, 2011. doi: 10.1109/ICRA.2011.5980391.

[157] Aishwarya Kamath, Mannat Singh, Yann LeCun, Ishan Misra, Gabriel Synnaeve, and Nicolas Carion. MDETR - modulated detection for end-to-end multi-modal understanding. *CoRR*, abs/2104.12763, 2021. URL https://arxiv.org/abs/2104.12763.

[158] Aishwarya Kamath, Mannat Singh, Yann André LeCun, Ishan Misra, Gabriel Synnaeve, and Nicolas Carion. MDETR - Modulated Detection for End-to-End Multi-Modal Understanding. In *Proc. ICCV*, 2021.

[159] Ivan Kapelyukh, Vitalis Vosylius, and Edward Johns. Dall-e-bot: Introducing web-scale diffusion models to robotics. *ArXiv*, abs/2210.02438, 2022.

[160] Ivan Kapelyukh, Vitalis Vosylius, and Edward Johns. Dall-e-bot: Introducing web-scale diffusion models to robotics. *IEEE Robotics and Automation Letters*, 2023.

[161] Andrej Karpathy and Li Fei-Fei. Deep Visual-semantic Alignments for Generating Image Descriptions. In *Proc. CVPR*, 2015.

[162] Pushkal Katara, Zhou Xian, and Katerina Fragkiadaki. Gen2sim: Scaling up robot learning in simulation with generative models. *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6672–6679, 2023.

[163] Sahar Kazemzadeh, Vicente Ordonez, Marc André Matten, and Tamara L. Berg. ReferItGame: Referring to Objects in Photographs of Natural Scenes. In

Proc. EMNLP, 2014.

[164] Tsung-Wei Ke, Nikolaos Gkanatsios, and Katerina Fragkiadaki. 3d diffuser actor: Policy diffusion with 3d scene representations. *arXiv preprint arXiv:2402.10885*, 2024.

[165] Tsung-Wei Ke, Nikolaos Gkanatsios, and Katerina Fragkiadaki. 3d diffuser actor: Policy diffusion with 3d scene representations, 2024.

[166] Justin Kerr, Chung Min Kim, Ken Goldberg, Angjoo Kanazawa, and Matthew Tancik. Lerf: Language embedded radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19729–19739, 2023.

[167] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models, 2019. URL https://arxiv.org/abs/1911.00172.

[168] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, P Fagan, Joey Hejna, Masha Itkina, Marion Lepert, Ye Ma, Patrick Tree Miller, Jimmy Wu, Suneel Belkhale, Shivin Dass, Huy Ha, Arhan Jain, Abraham Lee, Youngwoon Lee, Marius Memmel, Sung Yul Park, Ilija Radosavovic, Kaiyuan Wang, Albert Zhan, Kevin Black, Cheng Chi, Kyle Beltran Hatch, Shan Lin, Jingpei Lu, Jean-Pierre Mercat, Abdul Rehman, Pannag R. Sanketi, Archit Sharma, C. Blake Simpson, Quang Uyen Vuong, Homer Walke, Blake Wulfe, Ted Xiao, Jonathan Heewon Yang, Arefeh Yavary, Tony Zhao, Christopher Agia, Rohan Baijal, Mateo Guaman Castro, Da Ling Chen, Qiuyu Chen, Trinity Chung, Jaimyn Drake, Ethan P. Foster, Jensen Gao, David Antonio Herrera, Minho Heo, Kyle Hsu, Jiaheng Hu, Donovon Jackson, Charlotte Le, Yunshuang Li, Kevin Lin, Roy Lin, Zehan Ma, Abhiram Maddukuri, Suvir Mirchandani, Daniel Morton, Tony Nguyen, Abigail O'Neill, Rosa Maria Scalise, Derick Seale, Victor Son, Stephen Tian, Emi Tran, Andrew E. Wang, Yilin Wu, Annie Xie, Jingyun Yang, Patrick Yin, Yunchu Zhang, Osbert Bastani, Glen Berseth, Jeannette Bohg, Ken Goldberg, Abhinav Gupta, Abhishek Gupta, Dinesh Jayaraman, Joseph J. Lim, Jitendra Malik, Roberto Mart'in-Mart'in, Subramanian Ramamoorthy, Dorsa Sadigh, Shuran Song, Jiajun Wu, Michael C. Yip, Yuke Zhu, Thomas Kollar, Sergey Levine, and Chelsea Finn. Droid: A large-scale in-the-wild robot manipulation dataset. *ArXiv*, abs/2403.12945, 2024.

[169] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024.

[170] Sihyeon Kim, Sanghyeok Lee, Dasol Hwang, Jaewon Lee, Seong Jae Hwang, and Hyunwoo J Kim. Point cloud augmentation with weighted local transformations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 548–557, 2021.

[171] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[172] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[173] Scott Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, 34:975–986, 1984.

[174] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, et al. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017.

[175] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations. *International Journal of Computer Vision*, 123, 2016.

[176] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.

[177] Abhijit Kundu, Xiaoqi Yin, Alireza Fathi, David Ross, Brian Brewington, Thomas Funkhouser, and Caroline Pantofaru. Virtual multi-view fusion for 3d semantic segmentation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIV 16*, pages 518–535. Springer, 2020.

[178] Xin Lai, Jianhui Liu, Li Jiang, Liwei Wang, Hengshuang Zhao, Shu Liu, Xiaojuan Qi, and Jiaya Jia. Stratified transformer for 3d point cloud segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8500–8509, 2022.

[179] Xin Lai, Yuhui Yuan, Ruihang Chu, Yukang Chen, Han Hu, and Jiaya Jia. Mask-attention-free transformer for 3d instance segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3693–3703, 2023.

[180] Kuang-Huei Lee, Ofir Nachum, Mengjiao Yang, L. Y. Lee, Daniel Freeman,

Winnie Xu, Sergio Guadarrama, Ian S. Fischer, Eric Jang, Henryk Michalewski, and Igor Mordatch. Multi-game decision transformers. *ArXiv*, abs/2205.15241, 2022.

[181] Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. VisualBERT: A Simple and Performant Baseline for Vision and Language. *ArXiv*, abs/1908.03557, 2019.

[182] Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, et al. Grounded language-image pre-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10965–10975, 2022.

[183] Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, et al. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024.

[184] Xinghang Li, Minghuan Liu, Hanbo Zhang, Cunjun Yu, Jie Xu, Hongtao Wu, Chilam Cheang, Ya Jing, Weinan Zhang, Huaping Liu, et al. Vision-language foundation models as effective robot imitators. *arXiv preprint arXiv:2311.01378*, 2023.

[185] Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, et al. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024.

[186] Yang Li and Tatsuya Harada. Lepard: Learning partial point cloud matching in rigid and deformable scenes. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[187] Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B Tenenbaum, Antonio Torralba, and Russ Tedrake. Propagation networks for model-based control under partial observation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1205–1211. IEEE, 2019.

[188] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control, 2022. URL https://arxiv.org/abs/2209.07753.

[189] Zhihao Liang, Zhihao Li, Songcen Xu, Mingkui Tan, and Kui Jia. Instance segmentation in 3d scenes using semantic superpoint tree networks. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2763–2772, 2021.

[190] Bill Yuchen Lin, Chengsong Huang, Qian Liu, Wenda Gu, Sam Sommerer, and

Xiang Ren. On grounded planning for embodied tasks with language models, 2022. URL https://arxiv.org/abs/2209.00465.

[191] Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. Text2motion: From natural language instructions to feasible plans. *arXiv preprint arXiv:2303.12153*, 2023.

[192] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[193] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. URL http://arxiv.org/abs/1405.0312.

[194] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *Proc. ECCV*, 2014.

[195] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[196] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal Loss for Dense Object Detection. In *Proc. ICCV*, 2017.

[197] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

[198] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[199] Hao Liu, Lisa Lee, Kimin Lee, and Pieter Abbeel. Instruction-following agents with jointly pre-trained vision-language models. *arXiv preprint arXiv:2210.13431*, 2022.

[200] I-Chun Arthur Liu, Sicheng He, Daniel Seita, and Gaurav Sukhatme. Voxact-b: Voxel-based acting and stabilizing policy for bimanual manipulation. *CoRL*, 2024.

[201] Nan Liu, Shuang Li, Yilun Du, Joshua B. Tenenbaum, and Antonio Torralba. Learning to compose visual relations. *CoRR*, abs/2111.09297, 2021. URL https://arxiv.org/abs/2111.09297.

[202] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *CoRR*, abs/2107.13586, 2021. URL https://arxiv.org/abs/2107.13586.

[203] Shikun Liu, Stephen James, Andrew J Davison, and Edward Johns. Auto-lambda: Disentangling dynamic task relationships. *arXiv preprint arXiv:2202.03091*, 2022.

[204] Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. *arXiv preprint arXiv:2410.07864*, 2024.

[205] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. In *Proc. ECCV*, 2016.

[206] Weiyu Liu, Chris Paxton, Tucker Hermans, and Dieter Fox. Structformer: Learning spatial structure for language-guided semantic rearrangement of novel objects. *2022 International Conference on Robotics and Automation (ICRA)*, pages 6322–6329, 2021.

[207] Weiyu Liu, Chris Paxton, Tucker Hermans, and Dieter Fox. Structformer: Learning spatial structure for language-guided semantic rearrangement of novel objects. *arXiv preprint arXiv:2110.10189*, 2021.

[208] Weiyu Liu, Tucker Hermans, S. Chernova, and Chris Paxton. Structdiffusion: Object-centric diffusion for semantic rearrangement of novel objects. *ArXiv*, abs/2211.04604, 2022.

[209] Weiyu Liu, Tucker Hermans, Sonia Chernova, and Chris Paxton. Structdiffusion: Object-centric diffusion for semantic rearrangement of novel objects. *arXiv preprint arXiv:2211.04604*, 2022.

[210] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

[211] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, M. Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *ArXiv*, abs/1907.11692, 2019.

[212] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[213] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[214] Yongfei Liu, Xiangyi Zhang, Songyang Zhang, and Xuming He. Part-aware prototype network for few-shot semantic segmentation. In *European Conference on Computer Vision*, pages 142–158. Springer, 2020.

[215] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[216] Ze Liu, Zheng Zhang, Yue Cao, Han Hu, and Xin Tong. Group-Free 3D Object Detection via Transformers. In *Proc. ICCV*, 2021.

[217] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin transformer v2: Scaling up capacity and resolution, 2022.

[218] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[219] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[220] Guanxing Lu, Tengbo Yu, Haoyuan Deng, Season Si Chen, Yansong Tang, and Ziwei Wang. Anybimanual: Transferring unimanual policy for general bimanual manipulation. *arXiv preprint arXiv:2412.06779*, 2024.

[221] Jiahao Lu, Jiacheng Deng, Chuxin Wang, Jianfeng He, and Tianzhu Zhang. Query refinement transformer for 3d instance segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 18516–18526, 2023.

[222] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. ViLBERT: Pretraining Task-Agnostic Visiolinguistic Representations for Vision-and-Language Tasks. In *Proc. NeurIPS*, 2019.

[223] Jiasen Lu, Vedanuj Goswami, Marcus Rohrbach, Devi Parikh, and Stefan Lee. 12-in-1: Multi-Task Vision and Language Representation Learning. In *Proc. CVPR*, 2020.

[224] Qi Lv, Hao Li, Xiang Deng, Rui Shao, Yinchuan Li, Jianye Hao, Longxiang Gao, Michael Yu Wang, and Liqiang Nie. Spatial-temporal graph diffusion policy with kinematic modeling for bimanual robotic manipulation. *arXiv preprint arXiv:2503.10743*, 2025.

[225] Corey Lynch and Pierre Sermanet. Grounding language in play. *arXiv preprint arXiv:2005.07648*, 2020.

[226] Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data. *arXiv preprint arXiv:2005.07648*, 2020.

[227] Daoming Lyu, Fangkai Yang, Bo Liu, and Steven Gustafson. SDRL: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. *CoRR*, abs/1811.00090, 2018. URL http://arxiv.org/abs/1811.00090.

[228] Ajay Mandlekar, Fabio Ramos, Byron Boots, Li Fei-Fei, Animesh Garg, and Dieter Fox. IRIS: implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. *CoRR*, abs/1911.05321, 2019. URL http://arxiv.org/abs/1911.05321.

[229] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021.

[230] Puneet Mangla, Nupur Kumari, Abhishek Sinha, Mayank Singh, Balaji Krishnamurthy, and Vineeth N Balasubramanian. Charting the right manifold: Manifold mixup for few-shot learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

[231] Elman Mansimov, Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Generating images from captions with attention, 2015. URL https://arxiv.org/abs/1511.02793.

[232] Lucas Manuelli, Wei Gao, Peter Florence, and Russ Tedrake. kpam: Keypoint affordances for category-level robotic manipulation. In *Robotics Research: The 19th International Symposium ISRR*, pages 132–157. Springer, 2022.

[233] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *arXiv preprint arXiv:1904.12584*, 2019.

[234] Junhua Mao, Jonathan Huang, Alexander Toshev, Oana-Maria Camburu, Alan Loddon Yuille, and Kevin P. Murphy. Generation and Comprehension of Unambiguous Object Descriptions. In *Proc. CVPR*, 2016.

[235] Oier Mees, Lukas Hermann, and Wolfram Burgard. What matters in language conditioned robotic imitation learning over unstructured data. *IEEE Robotics and Automation Letters (RA-L)*, 7(4):11205–11212, 2022.

[236] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):

7327–7334, 2022.

[237] Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. Interactive furniture layout using interior design guidelines. *ACM transactions on graphics (TOG)*, 30(4):1–10, 2011.

[238] Toki Migimatsu and Jeannette Bohg. Object-centric task and motion planning in dynamic environments. *CoRR*, abs/1911.04679, 2019. URL `http://arxiv.org/abs/1911.04679`.

[239] So Yeon Min, Devendra Singh Chaplot, Pradeep Kumar Ravikumar, Yonatan Bisk, and Ruslan Salakhutdinov. Film: Following instructions in language with modular methods. In *International Conference on Learning Representations*, 2021.

[240] M. Minsky. A framework for representing knowledge. In R. J. Brachman and H. J. Levesque, editors, *Readings in Knowledge Representation*, pages 245–262. Kaufmann, Los Altos, CA, 1985. URL `http://web.media.mit.edu/~minsky/papers/Frames/frames.html`.

[241] Utkarsh A Mishra and Yongxin Chen. Reorientdiff: Diffusion model based reorientation for object manipulation. *arXiv preprint arXiv:2303.12700*, 2023.

[242] Ishan Misra, Rohit Girdhar, and Armand Joulin. An End-to-End Transformer Model for 3D Object Detection. In *Proc. ICCV*, 2021.

[243] Kaichun Mo, Shilin Zhu, Angel X. Chang, Li Yi, Subarna Tripathi, Leonidas J. Guibas, and Hao Su. PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[244] Igor Mordatch. Concept learning with energy-based models. *CoRR*, abs/1811.02486, 2018. URL `http://arxiv.org/abs/1811.02486`.

[245] Tomohiro Motoda, Ryo Hanai, Ryoichi Nakajo, Masaki Murooka, Floris Erich, and Yukiyasu Domae. Learning bimanual manipulation via action chunking and inter-arm coordination with transformers. *arXiv preprint arXiv:2503.13916*, 2025.

[246] Tongzhou Mu, Z. Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations. In *NeurIPS Datasets and Benchmarks*, 2021.

[247] Yao Mu, Tianxing Chen, Zanxin Chen, Shijia Peng, Zhiqian Lan, Zeyu Gao, Zhixuan Liang, Qiaojun Yu, Yude Zou, Mingkun Xu, Lunkai Lin, Zhiqiang Xie, Mingyu Ding, and Ping Luo. Robotwin: Dual-arm robot benchmark with generative digital twins, 2025. URL `https://arxiv.org/abs/2504.13059`.

[248] Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. *CoRR*, abs/1807.04742, 2018. URL http://arxiv.org/abs/1807.04742.

[249] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhi Gupta. R3m: A universal visual representation for robot manipulation. In *Conference on Robot Learning*, 2022.

[250] Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. *ArXiv*, abs/2406.02523, 2024.

[251] Khoi Nguyen and Sinisa Todorovic. Feature weighting and boosting for few-shot segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 622–631, 2019.

[252] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021.

[253] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. https://octo-models.github.io, 2023.

[254] Vicente Ordonez, Girish Kulkarni, and Tamara Berg. Im2text: Describing images using 1 million captioned photographs. In *Proc. NIPS*, 2011.

[255] Abhishek Padalkar, Acorn Pooley, Ajinkya Jain, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anika Singh, Anthony Brohan, Antonin Raffin, Ayzaan Wahid, Ben Burgess-Limerick, Beomjoon Kim, Bernhard Schölkopf, Brian Ichter, Cewu Lu, Charles Xu, Chelsea Finn, Chenfeng Xu, Cheng Chi, Chenguang Huang, Christine Chan, Chuer Pan, Chuyuan Fu, Coline Devin, Danny Driess, Deepak Pathak, Dhruv Shah, Dieter Büchler, Dmitry Kalashnikov, Dorsa Sadigh, Edward Johns, Federico Ceola, Fei Xia, Freek Stulp, Gaoyue Zhou, Gaurav S. Sukhatme, Gautam Salhotra, Ge Yan, Giulio Schiavi, Hao Su, Haoshu Fang, Haochen Shi, Heni Ben Amor, Henrik I Christensen, Hiroki Furuta, Homer Walke, Hongjie Fang, Igor Mordatch, Ilija Radosavovic, Isabel Leal, Jacky Liang, Jaehyung Kim, Jan Schneider, Jasmine Hsu, Jeannette Bohg, Jeff Bingham, Jiajun Wu, Jialin Wu, Jianlan Luo, Jiayuan Gu, Jie Tan, Jihoon Oh, Jitendra Malik, Jonathan Tompson, Jonathan Yang, Joseph J. Lim, João Silvério, Junhyek Han, Kanishka Rao, Karl Pertsch, Karol Hausman, Keegan Go, Keerthana Gopalakrishnan, Ken Goldberg, Kendra Byrne, Kenneth Oslund, Kento Kawaharazuka, Kevin Zhang, Keyvan Majd, Krishan Rana, Krishna Parasuram Srinivasan, Lawrence Yunliang Chen, Lerrel Pinto, Liam Tan, Lionel Ott, Lisa Lee, Masayoshi Tomizuka, Maximilian Du, Michael

Ahn, Mingtong Zhang, Mingyu Ding, Mohan Kumar Srirama, Mohit Sharma, Moo Jin Kim, Naoaki Kanazawa, Nicklas Hansen, Nicolas Manfred Otto Heess, Nikhil J. Joshi, Niko Suenderhauf, Norman Di Palo, Nur Muhammad Mahi Shafiullah, Oier Mees, Oliver Kroemer, Pannag R. Sanketi, Paul Wohlhart, Peng Xu, Pierre Sermanet, Priya Sundaresan, Quan Ho Vuong, Rafael Rafailov, Ran Tian, Ria Doshi, Russell Mendonca, Rutav Shah, Ryan Hoque, Ryan C. Julian, Samuel Bustamante, Sean Kirmani, Sergey Levine, Sherry Moore, Shikhar Bahl, Shivin Dass, Shuran Song, Sichun Xu, Siddhant Haldar, S. O. Adebola, Simon Guist, Soroush Nasiriany, Stefan Schaal, Stefan Welker, Stephen Tian, Sudeep Dasari, Suneel Belkhale, Takayuki Osa, Tatsuya Harada, Tatsuya Matsushima, Ted Xiao, Tianhe Yu, Tianli Ding, Todor Davchev, Tony Zhao, Travis Armstrong, Trevor Darrell, Vidhi Jain, Vincent Vanhoucke, Wei Zhan, Wenxuan Zhou, Wolfram Burgard, Xi Chen, Xiaolong Wang, Xinghao Zhu, Xuanlin Li, Yao Lu, Yevgen Chebotar, Yifan Zhou, Yifeng Zhu, Ying Xu, Yixuan Wang, Yonatan Bisk, Yoonyoung Cho, Youngwoon Lee, Yuchen Cui, Yueh hua Wu, Yujin Tang, Yuke Zhu, Yunzhu Li, Yusuke Iwasawa, Yutaka Matsuo, Zhuo Xu, and Zichen Jeff Cui. Open x-embodiment: Robotic learning datasets and rt-x models. *ArXiv*, abs/2310.08864, 2023.

[256] Aishwarya Padmakumar, Jesse Thomason, Ayush Shrivastava, Patrick Lange, Anjali Narayan-Chen, Spandana Gella, Robinson Piramuthu, Gokhan Tur, and Dilek Hakkani-Tur. Teach: Task-driven embodied agents that chat. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 2017–2025, 2022.

[257] Christopher R. Palmer and Christos Faloutsos. Density biased sampling: an improved method for data mining and clustering. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, page 82–92, New York, NY, USA, 2000. Association for Computing Machinery. ISBN 1581132174. doi: 10.1145/342009.335384. URL https://doi.org/10.1145/342009.335384.

[258] Priyam Parashar, Jay Vakil, Sam Powers, and Chris Paxton. Spatial-language attention policies for efficient robot learning. *arXiv preprint arXiv:2304.11235*, 2023.

[259] Simone Parisi, Aravind Rajeswaran, Senthil Purushwalkam, and Abhinav Kumar Gupta. The unsurprising effectiveness of pre-trained vision models for control. In *International Conference on Machine Learning*, 2022.

[260] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai,

and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

[261] Tim Pearce, Tabish Rashid, Anssi Kanervisto, Dave Bignell, Mingfei Sun, Raluca Georgescu, Sergio Valcarcel Macua, Shan Zheng Tan, Ida Momennejad, Katja Hofmann, and Sam Devlin. Imitating human behaviour with diffusion models, 2023.

[262] Songyou Peng, Kyle Genova, Chiyu Jiang, Andrea Tagliasacchi, Marc Pollefeys, Thomas Funkhouser, et al. Openscene: 3d scene understanding with open vocabularies. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 815–824, 2023.

[263] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[264] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

[265] Bryan A. Plummer, Liwei Wang, Christopher M. Cervantes, Juan C. Caicedo, Julia Hockenmaier, and Svetlana Lazebnik. Flickr30k Entities: Collecting Region-to-Phrase Correspondences for Richer Image-to-Sentence Models. In *Proc. ICCV*, 2015.

[266] Dean Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D.S. Touretzky, editor, *Proceedings of (NeurIPS) Neural Information Processing Systems*, pages 305 – 313. Morgan Kaufmann, December 1989.

[267] Dean A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3:88–97, 1991.

[268] Vitchyr H. Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. *CoRR*, abs/1903.03698, 2019. URL http://arxiv.org/abs/1903.03698.

[269] Mihir Prabhudesai, Mengning Wu, Amir Zadeh, Katerina Fragkiadaki, and Deepak Pathak. Diffusion beats autoregressive in data-constrained settings, 2025. URL https://arxiv.org/abs/2507.15857.

[270] Charles Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Proc. NIPS*, 2017.

[271] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages

652–660, 2017.

[272] Guocheng Qian, Yuchen Li, Houwen Peng, Jinjie Mai, Hasan Abed Al Kader Hammoud, Mohamed Elhoseiny, and Bernard Ghanem. Pointnext: Revisiting pointnet++ with improved training and scaling strategies. In *NeurIPS*, 2022.

[273] Guocheng Qian, Xingdi Zhang, Abdullah Hamdi, and Bernard Ghanem. Pix4point: Image pretrained transformers for 3d point cloud understanding. 2022.

[274] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C Lin. Scalable differentiable physics for learning and control. *arXiv preprint arXiv:2007.02168*, 2020.

[275] Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Yan Ding, Zhigang Wang, JiaYuan Gu, Bin Zhao, Dong Wang, et al. Spatialvla: Exploring spatial representations for visual-language-action model. *arXiv preprint arXiv:2501.15830*, 2025.

[276] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[277] Ilija Radosavovic, Tete Xiao, Stephen James, P. Abbeel, Jitendra Malik, and Trevor Darrell. Real-world robot learning with masked visual pre-training. *ArXiv*, abs/2210.03109, 2022.

[278] Ilija Radosavovic, Tete Xiao, Bike Zhang, Trevor Darrell, Jitendra Malik, and Koushil Sreenath. Learning humanoid locomotion with transformers. *ArXiv*, abs/2303.03381, 2023.

[279] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *CoRR*, abs/2102.12092, 2021. URL `https://arxiv.org/abs/2102.12092`.

[280] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *ArXiv*, abs/2204.06125, 2022.

[281] W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66:846–850, 1971.

[282] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction, 2021. URL `https://arxiv.org/abs/2103.13413`.

[283] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *Proc. CVPR*, 2016.

[284] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[285] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks. In *Proc. NIPS*, 2015.

[286] Moritz Reuss, Maximilian Li, Xiaogang Jia, and Rudolf Lioutikov. Goal-conditioned imitation learning using score-based diffusion policies. *arXiv preprint arXiv:2304.02532*, 2023.

[287] Moritz Reuss, Hongyi Zhou, Marcel Rühle, Ömer Erdinç Yağmurlu, Fabian Otto, and Rudolf Lioutikov. Flower: Democratizing generalist robot policies with efficient vision-language-action flow policies. In *7th Robot Learning Workshop: Towards Robots with Human-Level Abilities*, 2025.

[288] Seyed Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian D. Reid, and Silvio Savarese. Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression. In *Proc. CVPR*, 2019.

[289] Damien Robert, Bruno Vallet, and Loic Landrieu. Learning multi-view aggregation in the wild for large-scale 3d semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5575–5584, 2022.

[290] Junha Roh, Karthik Desingh, Ali Farhadi, and Dieter Fox. LanguageRefer: Spatial-Language Model for 3D Visual Grounding. In *Proc. CoRL*, 2021.

[291] Eric Rohmer, Surya PN Singh, and Marc Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ international conference on intelligent robots and systems*, pages 1321–1326. IEEE, 2013.

[292] Robin Rombach, A. Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10674–10685, 2022.

[293] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[294] David Rozenberszki, Or Litany, and Angela Dai. Language-grounded indoor 3d semantic segmentation in the wild. In *European Conference on Computer Vision*, pages 125–141. Springer, 2022.

[295] Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2018.

[296] Hyunwoo Ryu, Jiwoo Kim, Junwoo Chang, Hyun Seok Ahn, Joohwan Seo, Taehan Kim, Jongeun Choi, and Roberto Horowitz. Diffusion-edfs: Bi-equivariant denoising generative modeling on se (3) for visual robotic manipulation. *arXiv preprint arXiv:2309.02685*, 2023.

[297] Arka Sadhu, Kan Chen, and Ram Nevatia. Zero-shot grounding of objects from natural language queries. In *Pro. ICCV*, 2019.

[298] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L. Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, Seyedeh Sara Mahdavi, Raphael Gontijo Lopes, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. *ArXiv*, abs/2205.11487, 2022.

[299] Tim Salimans and Jonathan Ho. Should EBMs model the energy or the score? In *Energy Based Models Workshop - ICLR 2021*, 2021. URL `https://openreview.net/forum?id=9AS-TF2jRNb`.

[300] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1842–1850. JMLR.org, 2016.

[301] Gabriel Sarch, Yue Wu, Michael J Tarr, and Katerina Fragkiadaki. Open-ended instructable embodied agents with memory-augmented large language models. *arXiv preprint arXiv:2310.15127*, 2023.

[302] Jonas Schult, Francis Engelmann, Theodora Kontogianni, and Bastian Leibe. Dualconvmesh-net: Joint geodesic and euclidean convolutions on 3d meshes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8612–8622, 2020.

[303] Jonas Schult, Francis Engelmann, Alexander Hermans, Or Litany, Siyu Tang, and Bastian Leibe. Mask3d: Mask transformer for 3d semantic instance segmentation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8216–8223. IEEE, 2023.

[304] Daniel Seita, Pete Florence, Jonathan Tompson, Erwin Coumans, Vikas Sindhwani, Ken Goldberg, and Andy Zeng. Learning to rearrange deformable cables, fabrics, and bags with goal-conditioned transporter networks, 2021.

[305] Younggyo Seo, Danijar Hafner, Hao Liu, Fangchen Liu, Stephen James, Kimin Lee, and P. Abbeel. Masked world models for visual control. *ArXiv*, abs/2206.14244, 2022.

[306] Younggyo Seo, Junsup Kim, Stephen James, Kimin Lee, Jinwoo Shin, and P. Abbeel. Multi-view masked world models for visual robotic manipulation. *ArXiv*, abs/2302.02408, 2023.

[307] Nur Muhammad Mahi Shafiullah, Zichen Jeff Cui, Ariuntuya Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone, 2022.

[308] Piyush Sharma, Nan Ding, Sebastian Goodman, and Radu Soricut. Conceptual Captions: A Cleaned, Hypernymed, Image Alt-text Dataset for Automatic Image Captioning. In *Proc. ACL*, 2018.

[309] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations, 2018.

[310] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020.

[311] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, 2021.

[312] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Conference on Robot Learning*, pages 785–799. PMLR, 2023.

[313] Yawar Siddiqui, Lorenzo Porzi, Samuel Rota Bulò, Norman Müller, Matthias Nießner, Angela Dai, and Peter Kontschieder. Panoptic lifting for 3d scene understanding with neural fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9043–9052, 2023.

[314] Oana Sidi, Oliver Matias van Kaick, Yanir Kleiman, Hao Zhang, and Daniel Cohen-Or. Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering. *Proceedings of the 2011 SIGGRAPH Asia Conference*, 2011.

[315] Maximilian Sieb, Zhou Xian, Audrey Huang, Oliver Kroemer, and Katerina Fragkiadaki. Graph-structured visual imitation. In *Conference on Robot Learning*, pages 979–989. PMLR, 2020.

[316] Anthony Simeonov, Ankit Goyal, Lucas Manuelli, Lin Yen-Chen, Alina Sarmiento, Alberto Rodriguez, Pulkit Agrawal, and Dieter Fox. Shelving, stacking, hanging: Relational pose diffusion for multi-modal rearrangement. *arXiv preprint arXiv:2307.04751*, 2023.

[317] Sumeet Singh, Stephen Tu, and Vikas Sindhwani. Revisiting energy based models as policies: Ranking noise contrastive estimation and interpolating

energy models, 2023.

[318] Christian Smith, Yiannis Karayiannidis, Lazaros Nalpantidis, Xavi Gratal, Peng Qi, Dimos V Dimarogonas, and Danica Kragic. Dual arm manipulation—a survey. *Robotics and Autonomous systems*, 60(10):1340–1353, 2012.

[319] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/cb8da6767461f2812ae4290eac7cbc42-Paper.pdf.

[320] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015.

[321] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[322] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models, 2022.

[323] Elias Stengel-Eskin, Andrew Hundt, Zhuohong He, Aditya Murali, Nakul Gopalan, Matthew Gombolay, and Gregory Hager. Guiding multi-step rearrangement tasks with natural language instructions. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 1486–1501. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/stengel-eskin22a.html.

[324] Austin Stone, Ted Xiao, Yao Lu, Keerthana Gopalakrishnan, Kuang-Huei Lee, Quan Vuong, Paul Wohlhart, Brianna Zitkovich, Fei Xia, Chelsea Finn, and Karol Hausman. Open-world object manipulation using pre-trained vision-language models, 2023.

[325] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

[326] Francisco Suárez-Ruiz, Xian Zhou, and Quang-Cuong Pham. Can robots assemble an ikea chair? *Science Robotics*, 3(17):eaat6385, 2018.

[327] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.

[328] Chunyu Sun, Xin Tong, and Yang Liu. Semantic segmentation-assisted instance feature fusion for multi-level 3d part instance segmentation. *Computational Visual Media*, 2022.

[329] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H.S. Torr, and Timothy M. Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[330] Duy-Nguyen Ta, Eric Cousineau, Huihua Zhao, and Siyuan Feng. Conditional energy-based models for implicit policies: The gap between theory and practice, 2022.

[331] Ayça Takmaz, Elisabetta Fedele, Robert W Sumner, Marc Pollefeys, Federico Tombari, and Francis Engelmann. Openmask3d: Open-vocabulary 3d instance segmentation. *arXiv preprint arXiv:2306.13631*, 2023.

[332] Yang Tian, Sizhe Yang, Jia Zeng, Ping Wang, Dahua Lin, Hao Dong, and Jiangmiao Pang. Predictive inverse dynamics models are scalable learners for robotic manipulation, 2024. URL https://arxiv.org/abs/2412.15109.

[333] Zhuotao Tian, Hengshuang Zhao, Michelle Shu, Zhicheng Yang, Ruiyu Li, and Jiaya Jia. Prior guided feature enrichment network for few-shot segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 2020.

[334] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, page 1930–1936. AAAI Press, 2015. ISBN 9781577357384.

[335] Nikolaos Tsagkas, Oisin Mac Aodha, and Chris Xiaoxuan Lu. Vl-fields: Towards language-grounded neural implicit spatial representations. *arXiv preprint arXiv:2305.12427*, 2023.

[336] Yoshihisa Tsurumine and Takamitsu Matsubara. Goal-aware generative adversarial imitation learning from imperfect demonstration for robotic cloth manipulation, 2022.

[337] Hsiao-Yu Fish Tung, Ricson Cheng, and Katerina Fragkiadaki. Learning spatial common sense with geometry-aware recurrent networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2595–2603, 2019.

[338] Hsiao-Yu Fish Tung, Zhou Xian, Mihir Prabhudesai, Shamit Lal, and Katerina Fragkiadaki. 3d-oes: Viewpoint-invariant object-factorized environment simulators. *arXiv preprint arXiv:2011.06464*, 2020.

[339] Julen Urain, Niklas Funk, Georgia Chalvatzaki, and Jan Peters. Se (3)-diffusionfields: Learning cost functions for joint grasp and motion optimization through diffusion. *arXiv preprint arXiv:2209.03855*, 2022.

[340] Julen Urain, Niklas Funk, Jan Peters, and Georgia Chalvatzaki. Se (3)-

diffusionfields: Learning smooth cost functions for joint grasp and motion optimization through diffusion. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5923–5930. IEEE, 2023.

[341] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Duc Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1588–1597, 2019.

[342] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[343] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Proc. NIPS*, 2017.

[344] Thang Vu, Kookhoi Kim, Tung M Luu, Thanh Nguyen, and Chang D Yoo. Softgroup for 3d instance segmentation on point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2708–2717, 2022.

[345] Thang Vu, Kookhoi Kim, Tung Minh Luu, Xuan Thanh Nguyen, and Chang-Dong Yoo. Softgroup for 3d instance segmentation on point clouds. *ArXiv*, abs/2203.01509, 2022.

[346] Homer Walke, Kevin Black, Abraham Lee, Moo Jin Kim, Maximilian Du, Chongyi Zheng, Tony Zhao, Philippe Hansen-Estruch, Quan Ho Vuong, Andre Wang He, Vivek Myers, Kuan Fang, Chelsea Finn, and Sergey Levine. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning*, 2023.

[347] Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. Gensim: Generating robotic simulation tasks via large language models. *ArXiv*, abs/2310.01361, 2023.

[348] Peng-Shuai Wang. Octformer: Octree-based transformers for 3d point clouds. *arXiv preprint arXiv:2305.03045*, 2023.

[349] Sen Wang, Le Wang, Sanping Zhou, Jingyi Tian, Jiayi Li, Haowen Sun, and Wei Tang. Flowram: Grounding flow matching policy with region-aware mamba framework for robotic manipulation. *Conference on Computer Vision and Pattern Recognition*, 2025.

[350] Weiyue Wang, Ronald Yu, Qiangui Huang, and Ulrich Neumann. Sgpn: Similarity group proposal network for 3d point cloud instance segmentation. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2569–2578, 2018.

[351] Xiaogang Wang, Xun Sun, Xinyu Cao, Kai Xu, and Bin Zhou. Learning fine-grained segmentation of 3d shapes without part labels. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10271–10280, 2021.

[352] Xiaolong Wang, Yufei Ye, and Abhinav Gupta. Zero-shot recognition via semantic embeddings and knowledge graphs. *CoRR*, abs/1803.08035, 2018. URL http://arxiv.org/abs/1803.08035.

[353] Xin Wang, Thomas E Huang, Trevor Darrell, Joseph E Gonzalez, and Fisher Yu. Frustratingly simple few-shot object detection. *arXiv preprint arXiv:2003.06957*, 2020.

[354] Yushi Wang, Jonathan Berant, and Percy Liang. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1332–1342, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1129. URL https://aclanthology.org/P15-1129.

[355] Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*, 2022.

[356] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903, 2022. URL https://arxiv.org/abs/2201.11903.

[357] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. Citeseer, 2011.

[358] Hongtao Wu, Jikai Ye, Xin Meng, Chris Paxton, and Gregory Chirikjian. Transporters with visual foresight for solving unseen rearrangement tasks, 2022. URL https://arxiv.org/abs/2202.10765.

[359] Hongtao Wu, Ya Jing, Chilam Cheang, Guangzeng Chen, Jiafeng Xu, Xinghang Li, Minghuan Liu, Hang Li, and Tao Kong. Unleashing large-scale video generative pre-training for visual robot manipulation. *arXiv preprint arXiv:2312.13139*, 2023.

[360] Xiaoyang Wu, Yixing Lao, Li Jiang, Xihui Liu, and Hengshuang Zhao. Point transformer v2: Grouped vector attention and partition-based pooling. *Advances in Neural Information Processing Systems*, 35:33330–33342, 2022.

[361] Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers, 2022. URL https://arxiv.org/abs/2203.08913.

[362] Zhou Xian, Puttichai Lertkultanon, and Quang-Cuong Pham. Closed-chain manipulation of large objects by multi-arm robotic systems. *IEEE Robotics and Automation Letters*, 2(4):1832–1839, 2017.

[363] Zhou Xian, Shamit Lal, Hsiao-Yu Tung, Emmanouil Antonios Platanios, and Katerina Fragkiadaki. Hyperdynamics: Meta-learning object and agent dynamics with hypernetworks. *arXiv preprint arXiv:2103.09439*, 2021.

[364] Zhou Xian, Nikolaos Gkanatsios, Theophile Gervet, Tsung-Wei Ke, and Katerina Fragkiadaki. Chaineddiffuser: Unifying trajectory diffusion and keypose prediction for robotic manipulation. In *Conference on Robot Learning*, pages 2323–2339. PMLR, 2023.

[365] Zhou Xian, Bo Zhu, Zhenjia Xu, Hsiao-Yu Tung, Antonio Torralba, Katerina Fragkiadaki, and Chuang Gan. Fluidlab: A differentiable environment for benchmarking complex fluid manipulation. In *International Conference on Learning Representations*, 2023.

[366] Bin Xiao, Haiping Wu, Weijian Xu, Xiyang Dai, Houdong Hu, Yumao Lu, Michael Zeng, Ce Liu, and Lu Yuan. Florence-2: Advancing a unified representation for a variety of vision tasks, 2023. URL https://arxiv.org/abs/2311.06242.

[367] Tete Xiao, Ilija Radosavovic, Trevor Darrell, and Jitendra Malik. Masked visual pre-training for motor control. *ArXiv*, abs/2203.06173, 2022.

[368] Chenfeng Xu, Shijia Yang, Tomer Galanti, Bichen Wu, Xiangyu Yue, Bohan Zhai, Wei Zhan, Peter Vajda, Kurt Keutzer, and Masayoshi Tomizuka. Image2point: 3d point-cloud understanding with 2d image pretrained models. *arXiv preprint arXiv:2106.04180*, 2021.

[369] Danfei Xu, Roberto Martín-Martín, De-An Huang, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. Regression planning networks. *CoRR*, abs/1909.13072, 2019. URL http://arxiv.org/abs/1909.13072.

[370] Jiarui Xu, Sifei Liu, Arash Vahdat, Wonmin Byeon, Xiaolong Wang, and Shalini De Mello. Open-vocabulary panoptic segmentation with text-to-image diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2955–2966, 2023.

[371] Mengdi Xu, Yikang Shen, Shun Zhang, Yuchen Lu, Ding Zhao, Joshua B. Tenenbaum, and Chuang Gan. Prompting decision transformer for few-shot policy generalization. *ArXiv*, abs/2206.13499, 2022.

[372] Mengdi Xu, Yuchen Lu, Yikang Shen, Shun Zhang, Ding Zhao, and Chuang Gan. Hyper-decision transformer for efficient online policy adaptation. *ArXiv*, abs/2304.08487, 2023.

[373] Zhenjia Xu, Zhou Xian, Xingyu Lin, Cheng Chi, Zhiao Huang, Chuang Gan, and Shuran Song. Roboninja: Learning an adaptive cutting policy for multi-material objects. *arXiv preprint arXiv:2302.11553*, 2023.

[374] Karmesh Yadav, Ram Ramrakhya, Santhosh Kumar Ramakrishnan, Theo Gervet, John Turner, Aaron Gokaslan, Noah Maestre, Angel Xuan Chang, Dhruv Batra, Manolis Savva, et al. Habitat-matterport 3d semantics dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4927–4936, 2023.

[375] Boyu Yang, Chang Liu, Bohao Li, Jianbin Jiao, and Qixiang Ye. Prototype mixture models for few-shot semantic segmentation. In *European Conference on Computer Vision*, pages 763–778. Springer, 2020.

[376] Brian Yang, Huangyuan Su, Nikolaos Gkanatsios, Tsung-Wei Ke, Ayush Jain, Jeff Schneider, and Katerina Fragkiadaki. Diffusion-es: Gradient-free planning with diffusion for autonomous driving and zero-shot instruction following. *ArXiv*, abs/2402.06559, 2024.

[377] Yu-Qi Yang, Yu-Xiao Guo, Jian-Yu Xiong, Yang Liu, Hao Pan, Peng-Shuai Wang, Xin Tong, and Baining Guo. Swin3d: A pretrained transformer backbone for 3d indoor scene understanding. *arXiv preprint arXiv:2304.06906*, 2023.

[378] Yuyin Yang, Zetao Cai, Yang Tian, Jia Zeng, and Jiangmiao Pang. Gripper keypose and object pointflow as interfaces for bimanual robotic manipulation. *RSS*, 2025.

[379] Zhengyuan Yang, Boqing Gong, Liwei Wang, Wenbing Huang, Dong Yu, and Jiebo Luo. A fast and accurate one-stage approach to visual grounding. In *Proc. ICCV*, 2019.

[380] Zhengyuan Yang, Tianlang Chen, Liwei Wang, and Jiebo Luo. Improving one-stage visual grounding by recursive sub-query construction. In *Proc. ECCV*, 2020.

[381] Zhengyuan Yang, Songyang Zhang, Liwei Wang, and Jiebo Luo. SAT: 2D Semantics Assisted Training for 3D Visual Grounding. In *Proc. ICCV*, 2021.

[382] Lin Yen-Chen, Andy Zeng, Shuran Song, Phillip Isola, and Tsung-Yi Lin. Learning to see before learning to act: Visual pre-training for manipulation, 2021.

[383] Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Joshua B. Tenenbaum. Neural-symbolic VQA: disentangling reasoning from vision and language understanding. *CoRR*, abs/1810.02338, 2018. URL http://arxiv.org/abs/1810.02338.

[384] Kexin Yi, Chuang Gan, Yunzhu Li, Pushmeet Kohli, Jiajun Wu, Antonio

Torralba, and Joshua B. Tenenbaum. CLEVRER: CoLlision Events for Video REpresentation and Reasoning. In *Proc. ICLR*, 2020.

[385] Fenggen Yu, Kun Liu, Yan Zhang, Chenyang Zhu, and Kai Xu. Partnet: A recursive part decomposition network for fine-grained and hierarchical shape segmentation. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[386] Jiahui Yu, Yuanzhong Xu, Jing Yu Koh, Thang Luong, Gunjan Baid, Zirui Wang, Vijay Vasudevan, Alexander Ku, Yinfei Yang, Burcu Karagol Ayan, Ben Hutchinson, Wei Han, Zarana Parekh, Xin Li, Han Zhang, Jason Baldridge, and Yonghui Wu. Scaling autoregressive models for content-rich text-to-image generation, 2022. URL https://arxiv.org/abs/2206.10789.

[387] Lap Fai Yu, Sai Kit Yeung, Chi Keung Tang, Demetri Terzopoulos, Tony F Chan, and Stanley J Osher. Make it home: automatic optimization of furniture arrangement. *ACM Transactions on Graphics (TOG)-Proceedings of ACM SIGGRAPH 2011, v. 30,(4), July 2011, article no. 86*, 30(4), 2011.

[388] Licheng Yu, Patrick Poirson, Shan Yang, Alexander C. Berg, and Tamara L. Berg. Modeling Context in Referring Expressions. In *Proc. ECCV*, 2016.

[389] Tianhe Yu, Ted Xiao, Austin Stone, Jonathan Tompson, Anthony Brohan, Su Wang, Jaspiar Singh, Clayton Tan, Meredith Dee, Jodilyn Peralta, Brian Ichter, Karol Hausman, and F. Xia. Scaling robot learning with semantically imagined experience. *ArXiv*, abs/2302.11550, 2023.

[390] Zhihao Yuan, Xu Yan, Yinghong Liao, Ruimao Zhang, Zhen Li, and Shuguang Cui. InstanceRefer: Cooperative Holistic Understanding for Visual Grounding on Point Clouds through Instance Multi-level Contextual Referring. In *Proc. ICCV*, 2021.

[391] Yanjie Ze, Ge Yan, Yueh-Hua Wu, Annabella Macaluso, Yuying Ge, Jianglong Ye, Nicklas Hansen, Li Erran Li, and Xiaolong Wang. Gnfactor: Multi-task real robot learning with generalizable neural feature fields. *arXiv preprint arXiv:2308.16891*, 2023.

[392] Yanjie Ze, Zixuan Chen, Wenhao Wang, Tianyi Chen, Xialin He, Ying Yuan, Xue Bin Peng, and Jiajun Wu. Generalizable humanoid manipulation with improved 3d diffusion policies. *arXiv preprint arXiv:2410.10803*, 2024.

[393] Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2024.

[394] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani,

et al. Transporter networks: Rearranging the visual world for robotic manipulation. *arXiv preprint arXiv:2010.14406*, 2020.

[395] Andy Zeng, Peter R. Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, and Johnny Lee. Transporter networks: Rearranging the visual world for robotic manipulation. In *Conference on Robot Learning*, 2020.

[396] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, et al. Transporter networks: Rearranging the visual world for robotic manipulation. In *Conference on Robot Learning*, pages 726–747. PMLR, 2021.

[397] Biao Zhang and Peter Wonka. Point cloud instance segmentation using probabilistic embeddings. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8879–8888, 2021.

[398] Fan Zhang and Michael Gienger. Affordance-based robot manipulation with flow matching. *arXiv preprint arXiv:2409.01083*, 2024.

[399] Kevin Zhang, Mohit Sharma, Jacky Liang, and Oliver Kroemer. A modular robotic arm control stack for research: Franka-interface and frankapy. *arXiv preprint arXiv:2011.02398*, 2020.

[400] Qinglun Zhang, Zhen Liu, Haoqiang Fan, Guanghui Liu, Bing Zeng, and Shuaicheng Liu. Flowpolicy: Enabling fast and robust 3d flow-based policy via consistency flow matching for robot manipulation. *arXiv preprint arXiv:2412.04987*, 2024.

[401] Renrui Zhang, Ziyu Guo, Wei Zhang, Kunchang Li, Xupeng Miao, Bin Cui, Yu Qiao, Peng Gao, and Hongsheng Li. Pointclip: Point cloud understanding by clip. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8552–8562, 2022.

[402] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5628–5635. IEEE, 2018.

[403] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 16259–16268, 2021.

[404] Lichen Zhao, Daigang Cai, Lu Sheng, and Dong Xu. 3DVG-Transformer: Relation Modeling for Visual Grounding on Point Clouds. In *Proc. ICCV*, 2021.

[405] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *RSS*, 2023.

[406] Weiguang Zhao, Yuyao Yan, Chaolong Yang, Jianan Ye, Xi Yang, and Kaizhu Huang. Divide and conquer: 3d point cloud instance segmentation with point-wise binarization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 562–571, October 2023.

[407] Jia Zheng, Junfei Zhang, Jing Li, Rui Tang, Shenghua Gao, and Zihan Zhou. Structured3d: A large photo-realistic dataset for structured 3d modeling. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IX 16*, pages 519–535. Springer, 2020.

[408] Zhisheng Zhong, Jiequan Cui, Yibo Yang, Xiaoyang Wu, Xiaojuan Qi, Xiangyu Zhang, and Jiaya Jia. Understanding imbalanced semantic segmentation through neural collapse. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19550–19560, 2023.

[409] Ziyuan Zhong, Davis Rempe, Danfei Xu, Yuxiao Chen, Sushant Veer, Tong Che, Baishakhi Ray, and Marco Pavone. Guided conditional diffusion for controllable traffic simulation. *ArXiv*, abs/2210.17366, 2022.

[410] Xingyi Zhou, Rohit Girdhar, Armand Joulin, Phillip Krähenbühl, and Ishan Misra. Detecting twenty-thousand classes using image-level supervision, 2022. URL `https://arxiv.org/abs/2201.02605`.

[411] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks, 2020.

[412] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: Deformable Transformers for End-to-End Object Detection. In *Proc. ICLR*, 2021.

[413] Xupeng Zhu, Dian Wang, Ondrej Biza, Guanang Su, Robin Walters, and Robert Platt. Sample efficient grasp learning using equivariant models, 2022.

[414] Yifeng Zhu, Jonathan Tremblay, Stan Birchfield, and Yuke Zhu. Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs. *CoRR*, abs/2012.07277, 2020. URL `https://arxiv.org/abs/2012.07277`.

[415] Xueyan Zou, Zi-Yi Dou, Jianwei Yang, Zhe Gan, Linjie Li, Chunyuan Li, Xiyang Dai, Harkirat Behl, Jianfeng Wang, Lu Yuan, et al. Generalized decoding for pixel, image, and language. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15116–15127, 2023.