UC Berkeley
UC Berkeley Electronic Theses and Dissertations

Title

General Robot Manipulation with Multi-Modal Vision-Language-Action Models

Permalink

https://escholarship.org/uc/item/1nn2p5gb

ISBN

9798173787262

Author

Black, Kevin

Publication Date

2026-08-10

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

General Robot Manipulation with Multi-Modal Vision-Language-Action Models

by

Kevin Black

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Sergey Levine, Chair
Professor Sewon Min
Professor Shuran Song

Summer 2026

General Robot Manipulation with Multi-Modal Vision-Language-Action Models

Copyright 2026

by

Kevin Black

Abstract

General Robot Manipulation with Multi-Modal Vision-Language-Action Models

by

Kevin Black

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Sergey Levine, Chair

Robot learning holds tremendous potential to bridge the gap between artificial intelligence and the physical world. However, achieving human-level dexterous manipulation across a wide variety of robots, environments, and tasks remains a great challenge. Often, this is blamed on a dearth of robot training data, but that situation is quickly changing. Spurred by the tremendous successes of digital AI agents, robotics companies and academic labs alike have increasingly expanded the scale of robot data collection. In this dissertation, we explore how to best use that data to train and deploy general-purpose robot manipulation policies; much like how the large-scale pre-training of language models sparked a revolution in cognitive labor, we aim to do the same for physical labor. We begin by proposing a full training system for a generalist robot policy, including a model architecture that is designed for the particular needs of generalist robotics. We train a series of models of different sizes on a highly heterogeneous amalgamation of datasets, and study the behavior of large-scale robot policy pre-training. Then, we address the challenge of real-time control with big, compute-intensive robot policies; we devise an algorithm to execute such policies asynchronously and smoothly, showing that our algorithm greatly improves performance in the presence of inference latency. Lastly, we propose a hierarchical system that can directly leverage Internet-scale non-robot video data to improve its semantic and visual reasoning, and find that this system exhibits surprising generalization capabilities even with relatively little robot data.

To my parents

Contents

Contents **ii**

List of Figures **v**

List of Tables **xvii**

1 Introduction **1**
1.1 Publications 4

I Training a Generalist Robot Policy **6**

2 Octo: A Generalist Robot Policy **7**
2.1 Introduction 7
2.2 Related Work 9
2.3 The Octo Model 11
2.4 Experiments 14
2.5 Discussion 19

3 π_0: A Vision-Language-Action Flow Model for General Robot Control **23**
3.1 Introduction 23
3.2 Related Work 26
3.3 Overview 28
3.4 The π_0 Model 29
3.5 Data Collection and Training Recipe 30
3.6 Experimental Evaluation 34
3.7 Discussion, Limitations, and Future Work 44

II Real-Time Control with Large Generalist Policies **47**

4 Real-Time Chunking **48**
4.1 Introduction 49

4.2 Related Work . . . 50
4.3 Preliminaries and Motivation . . . 52
4.4 Real-Time Chunking via Inpainting . . . 54
4.5 Experiments . . . 57
4.6 Discussion . . . 62

5 Training-Time Action Conditioning for Efficient Real-Time Chunking 63
5.1 Introduction . . . 63
5.2 Related Work . . . 64
5.3 Preliminaries . . . 64
5.4 Training-Time Action Conditioning . . . 65
5.5 Experiments . . . 67
5.6 Discussion and Future Work . . . 69

III Policy Generalization via World Models 71

6 SuSIE: Subgoal Synthesis via Image Editing 72
6.1 Introduction . . . 72
6.2 Related Work . . . 73
6.3 Preliminaries and Problem Statement . . . 76
6.4 SuSIE: Subgoal Synthesis via Image Editing . . . 76
6.5 Experimental Evaluation . . . 78
6.6 Discussion . . . 86

7 $\pi_{0.7}$: a Steerable Generalist Robotic Foundation Model 89
7.1 Introduction . . . 89
7.2 Related Work . . . 92
7.3 Flow-Based Vision-Language-Action Models . . . 93
7.4 $\pi_{0.7}$ Overview . . . 94
7.5 Diversifying the Prompt . . . 95
7.6 The $\pi_{0.7}$ Model and Training Recipe . . . 98
7.7 Prompting $\pi_{0.7}$ at runtime . . . 100
7.8 Robot system details . . . 101
7.9 Experimental Evaluation . . . 103
7.10 Discussion . . . 117

8 Conclusion 119

Bibliography 122

A Appendicies for Chapter 2 147
A.1 Contributions . . . 147

A.2 Octo Code Example . . . 148
A.3 Data mixture . . . 148
A.4 Training Hyperparameters . . . 148
A.5 Things that Worked and Did Not Work (Yet) . . . 150
A.6 Experimental Setups . . . 152

B Appendices for Chapter 3 **156**
B.1 Contributions . . . 156
B.2 Model Architecture Details . . . 157
B.3 Non-VLM Baseline Architecture . . . 158
B.4 Inference . . . 159
B.5 Evaluation Details . . . 160

C Appendices for Chapter 4 **163**
C.1 Broader Impacts . . . 163
C.2 The Necessity of Guidance Weight Clipping (β) . . . 163
C.3 Latency Measurements . . . 165
C.4 Additional Simulated Ablations . . . 165
C.5 Hyperparameters . . . 167
C.6 Code Release . . . 167
C.7 Compute Resources . . . 167

D Appendices for Chapter 5 **168**

E Appendices for Chapter 6 **170**
E.1 Implementation Details . . . 170
E.2 Experimental Setup . . . 174
E.3 Qualtitative Examples of Video Generation . . . 175
E.4 Qualitative Examples of SuSIE on CALVIN . . . 178

F Appendices for Chapter 7 **179**
F.1 Contributions . . . 179
F.2 Attention pattern . . . 180
F.3 Training of the world model . . . 180
F.4 Inference speed and optimization . . . 180
F.5 Comparison of action spaces in cross-embodiment transfer . . . 182
F.6 Human subject study for cross-embodiment shirt folding . . . 182
F.7 Detailed task descriptions and scoring rubric . . . 184

List of Figures

2.1 We introduce Octo, an open-source, generalist policy for robotic manipulation. Octo is a transformer-based policy pretrained on 800k diverse robot episodes from the Open X-Embodiment dataset [240]. It supports flexible task and observation definitions and can be quickly finetuned to new observation and action spaces. . 8

2.2 **Model architecture. Left**: Octo tokenizes task descriptions (green) and input observations (blue) using a pretrained language model and a lightweight CNN, respectively. **Top**: The transformer backbone processes the sequence of task and observation tokens and produces readout tokens (purple) that get passed to output heads to produce actions. **Bottom**: The block-wise attention structure of the transformer allows us to add or remove inputs and outputs during finetuning: for example, we can add new observations (blue, dashed) or action spaces (purple, dashed) without modifying any pretrained parameters. 10

2.3 **Training dataset composition.** We curate a subset of 25 datasets from the Open X-Embodiment dataset that have image observations, end-effector actions, and show diverse behaviors. The pie chart visualizes the fractions that each dataset contributes to every training batch on average. The dataset weights are determined by the number of samples in each dataset with small modifications to balance dataset size and diversity (see Section 2.3 for details). 20

2.4 **Evaluation Tasks.** We evaluate Octo on 9 real robot setups across 4 institutions. Our evaluations capture diverse object interactions (e.g., "WidowX BridgeV2"), long task horizons (e.g., "Stanford Coffee") and precise manipulation (e.g., "Berkeley Peg Insertion"). We evaluate Octo's capabilities to control robots in environments from the pretraining data out-of-the-box and to efficiently finetune to new tasks and environments with small target domain datasets. We also test finetuning with new observations (force-torque inputs for "Berkeley Peg Insertion"), action spaces (joint position control in "Berkeley Pick-Up" and "Berkeley Bimanual") and new robot embodiments (e.g., "Berkeley Bimanual" and "Berkeley Coke"). 21

2.5 **Zero-Shot Evaluation.** Out-of-the-box, Octo can control multiple robots in environments from the pretraining data. When using natural language to specify tasks, Octo outperforms RT-1-X [240], the current best openly available generalist robot policy across three different robot embodiments and setups. Octo also performs similarly to RT-2-X [387] on the tested WidowX and RT-1 Robot tasks.[1] 21

2.6 **Model Scaling.** The performance of Octo improves with larger model sizes on both UR5 and WidowX tasks. Success rates are averaged over 10 trials on one language-conditioned task per robot. 22

3.1 Our generalist robot policy uses a pre-trained vision-language model (VLM) backbone, as well as a diverse cross-embodiment dataset with a variety of dexterous manipulation tasks. The model is adapted to robot control by adding a separate *action expert* that produces continuous actions via flow matching, enabling precise and fluent manipulation skills. The model can then be used directly to perform tasks based on a prompt, or fine-tuned on high-quality data to enable complex multi-stage tasks, such as folding multiple articles of laundry or assembling a box. 24

3.2 **π_0 controls a mobile manipulator to fold laundry.** Our model is pre-trained on diverse data from 7 distinct robot configurations and 68 tasks, and can then either be prompted directly or fine-tuned to complex downstream tasks, as in the case of this laundry folding policy, which fetches laundry from a dryer, packs it into a hamper, brings the hamper to a folding table, and then folds each article of clothing. 25

3.3 **Overview of our framework.** We start with a pre-training mixture, which consists of both our own dexterous manipulation datasets and open-source data. We use this mixture to train our flow matching VLA model, which consists of a larger VLM backbone and a smaller *action expert* for processing robot states and actions. The VLM backbone weights are initialized from PaliGemma [19], providing representations learned from large-scale Internet pre-training. The resulting π_0 model can be used to control multiple robot embodiments with differing action spaces to accomplish a wide variety of tasks. 27

3.4 **Overview of our dataset**: The pre-training mixture consists of a subset of OXE [65] and the π dataset. We use a subset of OXE, which we refer to as OXE Magic Soup [161]. The right figure illustrates the weight of the different datasets in the pre-training mixture. The left figure illustrates their relative sizes as measured by the number of steps. 31

3.5 **The robots used in our experiments.** These include single and dual-arm manipulators with 6-DoF and 7-DoF arms, as well as holonomic and nonholonomic mobile manipulators. π_0 is trained jointly on all of these platforms. 33

3.6 **Out-of-box evaluation tasks**: To evaluate our base model, we run it after pre-training on five tasks: **shirt folding**, **bussing easy**, **bussing hard**, **grocery bagging**, and **toast out of toaster**. The tasks require a combination of dexterous manipulation, multi-stage behaviors, and semantic recognition. 35

3.7 **Out-of-box evaluation results**: We evaluate π_0 trained for the full 700k steps, a version trained for 160k steps that matches the number of updates for baseline models, π_0-small, and three baselines: OpenVLA and Octo trained on all of our data, and OpenVLA trained only on the UR5e tasks (which we found to work better on UR5e tasks). Across all tasks and all comparisons, even the "parity" version of our model outperforms all baselines, and the full version of our model achieves the best results by a large margin. 37
3.8 **The tasks in our language evaluation.** We evaluate our model on 3 different language-conditioned tasks, each of which requires following a sequence of intermediate language commands. The tasks involve bussing a table (top) to put dishes in a bin and garbage in a trash bin, setting a table (middle) by taking items out of a bin, and packing a shopping bag (bottom). 38
3.9 **Language evaluation.** We compare "flat" versions of our policies, $-$`flat`, which receive only the overall task command (e.g., "bag the groceries") with a method that receives intermediate commands from a human expert, $-$`human`, or a high-level VLM policy, $-$`HL`. We also compare our model to a small non-VLM variant under the "expert" condition, π_0 and π_0-small, in terms of language following accuracy. The results show a significant improvement with π_0 from intermediate language commands provided by a human expert and to a lesser degree by an autonomous high-level policy. Notably, due to π_0-small's limited language following ability, overall it does not gain with the addition of a high-level expert. 39
3.10 **Fine-tuning evaluation tasks**: We fine-tune our model to a variety of downstream tasks that are distinct from the tasks seen in pre-training. Our tasks represent a range of similarity from the pre-training tasks, with tasks that are most similar to pre-training (**stack bowls** and **towel folding**), a task that introduces an unseen new element (a microwave), and tasks that require new motions and new object types (**Franka items in drawer** and **paper towel replacement**). 40
3.11 **Fine-tuning with varying amounts of data.** π_0 can learn some easier tasks even with smaller amounts of data, and the pre-trained model often attains a larger improvement over the model trained from scratch. 41
3.12 **We evaluate a range of complex and temporally extended tasks.** This includes: folding laundry from a bin with a stationary (a) or mobile (b) robot, bussing a real lunch table (c), assembling a box (d), packing eggs into a carton (e), and packing food into a to-go box (f). These tasks require combining dozens of individual behaviors, such as grasping, stacking, folding, and flattening, generalization to a huge variety of object configurations, and complex physical properties, such as deformable objects or flexible cardboard. 43
3.13 **Post-training results on complex tasks** in terms of average scores over 10 trials. The full pre-trained π_0 model attains more than 50% of the maximum score across all of the tasks, and typically outperforms the ablations, with especially significant improvements on the hardest tasks. 46

4.1 **Top:** Real-time chunking (RTC) enables the robot to perform highly dexterous and dynamic tasks, such as lighting a match—even in the presence of inference delays in excess of 300 milliseconds, corresponding to more than 30% of the model's prediction horizon. **Bottom:** RTC performs the same robot motion 20% faster than synchronous inference [32, 161, 44, 134, 159, 315], and smoother than all competing methods, including temporal ensembling [375]. The shown positions, velocities, and accelerations correspond to the shoulder joint of one arm, and are taken from the first 10 seconds of a real autonomous match-lighting rollout. . . 48

4.2 An illustration of a typical bifurcation between consecutive chunks. Inference is started between timesteps 3 and 4. The original chunk that was executing, $\{a_t\}$ (black), had planned to go above the obstacle while the newly generated chunk $\{a'_t\}$ (red) goes below the obstacle. However, $\{a'_t\}$ is not available until $d = 7$ steps later. A naive asynchronous algorithm might jump from a_{10} to a'_{11}, inducing a very high, out-of-distribution acceleration. Temporal ensembling [375], i.e., interpolating between chunks, reduces the acceleration but produces poor actions. 53

4.3 A diagram illustrating how action generation attends to the previous action chunk in real-time chunking. If inference starts after the execution of a_{-1} and the inference delay is $d = 4$, then the newly generated chunk will not be available until after a_3 is consumed. Therefore, $a_{0:3}$ are "frozen" and are attended to with a full guidance weight of 1. In the intermediate region, $a_{4:10}$, actions from the previous chunk are available but may be updated, since inference will have finished before a_4 is needed. This region is attended to with an exponentially decreasing guidance weight. Finally, the last $s = 5$ actions are beyond the end of the previous chunk, and need to be freshly generated. The execution horizon, s, is a hyperparameter constrained by $d \leq s \leq H - d$. 54

4.4 A comparison of naive inpainting (hard masking) and our proposed soft masking method: note that hard masking does not match the frozen region very well and produces faster changes in direction. 56

4.5 **Top left:** Kinetix environments; each involves getting a green object on the left to touch a blue one on the right. **Bottom left:** Execution horizon vs. solve rate with a fixed inference delay of 1. Only RTC and BID take full advantage of faster updates, showing strictly increasing performance with decreasing execution horizon. **Right:** Inference delay vs. solve rate with a fixed execution horizon of $s = \max(d, 1)$. RTC outperforms all baselines. Furthermore, soft masking (Sec. 4.4) improves performance at lower inference delays and execution horizons. Each data point represents 2048 trials, and 95% Wilson score intervals are shaded in. 59

4.6 **Top:** Controller steps (equivalent to elapsed time with inference pauses removed multiplied by 50Hz) vs. cumulative progress for each task, aggregated across all delays. Progress is measured in discrete steps corresponding to the subsections of each task. **Left:** Time (including inference pauses) vs. cumulative progress aggregated across all tasks. The x-axis is log scale to better show progress during both short and long-horizon tasks. **Right:** Inference delay vs. average throughput, defined as the proportion of task completed divided by duration of episode averaged over episodes. Error bars are ±1 SEM. Average throughput gives a balanced view of both speed and performance for each method. Neither TE variant can run at +100 or +200ms of injected latency, causing such high oscillations that the robot's protective stop is triggered. 61

5.1 A diagram illustrating two overlapping action chunks. The d actions between t and $t+d$, taken from the previous chunk, are the action prefix (red). From the diagram, we can easily see that we must satisfy the constraint $t+d \leq t-s+H \rightarrow d \leq H-s$ to have a valid action prefix. Note that inference-time RTC uses all $H-s$ overlapping actions (red and yellow) to guide the generation of the current chunk, whereas training-time RTC only uses the first d actions (red). 65

5.2 An illustration of our conditioning architecture, as applied to a standard diffusion transformer such as the $\pi_{0.6}$ action expert. We always feed in ground-truth, non-noisy prefix actions, while learning to denoise the postfix actions. The flow matching timestep differs between tokens, which indicates the inference delay to the model. 66

5.3 Simulated results: inference delay vs. solve rate with a fixed execution horizon of $s = \max(d, 1)$. Training-time RTC performs better than inference-time RTC at inference delays of 2 or higher. Each data point represents 2048 trials, and 95% Wilson score intervals are shaded in. 67

5.4 Real-world evaluation tasks: building a cardboard box and making espresso (including grinding, tamping, extracting, and pouring). 68

5.5 Real-world results: success rate and duration for espresso making and box building. Training-time and inference-time RTC perform similarly, while both improving speed over synchronous inference. Error bars represent 68% Wilson score intervals for success rate and ±1 SEM for duration. 70

6.1 **(SuSIE)** Our method leverages a pretrained image-editing model to generate future subgoals based on language commands. A low-level goal-reaching policy then executes the actions needed to reach each subgoal. Alternating this loop enables us to solve the task. 74

6.2 **(Experimental setup) (L)** We evaluate our method on 9 tasks across 3 real-world scenes. In terms of semantic generalization, the scenes become progressively more difficult, due to both an increasing visual departure from the robot training data and an increasingly confounding mixture of both seen and unseen objects. **(R)** In simulation, we evaluate our method in the zero-shot setting of the CALVIN benchmark, which involves training on 3 environments (A, B, and C) and testing on a 4th environment (D). The environments differ in table texture, positioning of furniture elements, and possible configurations of various colored blocks. Each environment comes with 34 language-specified tasks. 79

6.3 **(Visualizing rollouts)** Visualized rollouts from SuSIE, RT-2-X, and Oracle GCBC on the task "put the yellow bell pepper in the ceramic bowl" from Scene B. While RT-2-X and Oracle GCBC fail to grasp the object, the generated subgoals from SuSIE precisely guide the low-level controller, improving low-level skill execution for difficult manipulation tasks. 86

6.4 **(Comparison of subgoal quality)** A comparison between 3 subgoal generation models: one trained on both robot and video data (Bridge + Smthn-Smthn), used throughout this chapter; one trained on robot data only (Bridge Only); and one trained without InstructPix2Pix initialization (From Scratch). The top half (In Distribution) comes from the BridgeData V2 validation set, while the bottom half (Out of Distribution) comes from our evaluation Scenes B and C, unseen in BridgeData V2. Both Internet-scale pretraining and video cotraining and important for generating high-quality subgoals, especially in the zero-shot generalization setting. 87

7.1 We introduce $\pi_{0.7}$, a steerable generalist robot foundation model that can perform dexterous tasks across many tasks, environments, and robots. $\pi_{0.7}$ is trained with diverse prompts that contain not only the task description, but detailed language, generated subgoal images, and episode metadata. This provides richer context about not only *what* to do, but also *how* to do it, making it possible for $\pi_{0.7}$ to leverage broad set of both robot and non-robot data and compose the skills in this data in new ways to solve new tasks. 90

7.2 **Architecture overview.** The $\pi_{0.7}$ model is a 5B-parameter VLA consisting of a 4B VLM backbone, a `MEM`-style video history encoder and a 860M parameter action expert. The model's context includes multiple distinct modalities, including language commands, episode metadata that describes the data quality and strategy, and multimodal inputs such as subgoal images. At runtime, the language commands are produced by a high-level semantic policy based on the same architecture, and the subgoal images are produced by a lightweight world model based on the BAGEL image generation model [72]. 94

7.3 **Prompt overview.** $\pi_{0.7}$ uses diverse modalities of context in the prompt, including: subtask instructions, subgoal images, and episode metadata. We train the model with dropout for each component, and then prompt the model flexibly combining modalities. For example, when using the UR5e bimanual manipulator to fold a shirt, we use subgoal image and metadata prompting. 97
7.4 **Illustrations of some of the robots in our experiments.** We evaluate $\pi_{0.7}$ on a variety of robots, including bimanual mobile manipulators (left), static bimanual robots (middle), and a bimanual UR5e setup (right) that we use for cross-embodiment experiments. 102
7.5 **Illustration of selected evaluation tasks.** We evaluate $\pi_{0.7}$ on a number of tasks, and two of the more longer-horizon ones are visualized here. For some tasks such as "Take Out Trash", we provide a coarse instruction like "take out the trash" and $\pi_{0.7}$ performs the full long-horizon task. For other tasks which do not appear in the training data for $\pi_{0.7}$ such as "Toasting a Bagel", we can leverage the strong language following capabilities of $\pi_{0.7}$ to coach it to perform the task with a series of detailed instructions that break down the task step-by-step. 103
7.6 **Out-of-the-box dexterity:** $\pi_{0.7}$ can perform a wide range of highly dexterous tasks directly out of the box. We consider tasks from $\pi^*_{0.6}$ [317] (top row) and a number of other dexterous tasks including ones from the "Robot Olympics" experiments (bottom row). For the tasks from $\pi^*_{0.6}$, we report success rate and normalized throughput (relative to the specialist model; raw throughput means successes per hour), while for other tasks we report task progress. We find that the same $\pi_{0.7}$ model can match the performance of the task-specific post-trained specialist policy from $\pi^*_{0.6}$ or $\pi_{0.6}$ for each of these tasks, and even achieve higher throughput than the RL specialists in diverse laundry folding and box building. 104
7.7 **Impact of prompt composition and evaluation data on out-of-the-box performance:** We compare $\pi_{0.7}$ with two ablations: one that does not include episode metadata in the context, $\pi_{0.7}$ (no metadata), and another that does not include data from autonomous evaluation episodes during training, $\pi_{0.7}$ (no eval data). We find that $\pi_{0.7}$ outperforms both $\pi_{0.7}$ (no metadata) and $\pi_{0.7}$ (no eval data) across the board, with the gap most prominent in throughput. Throughput (successes/hour) here is normalized relative to $\pi_{0.7}$. 105
7.8 **Tasks that require memory:** $\pi_{0.7}$ can also perform tasks that require explicitly keeping track of prior context, achieving similar or better performance compared to the specialist policies with memory fine-tuned to some of the tasks in the MEM paper [322]. 105

7.9 **Broad instruction following in novel environments:** We evaluate $\pi_{0.7}$ on 14 instruction following scenarios, each of which involve following a sequence of 3-6 open-ended instructions, across 4 unseen kitchen and 2 unseen bedroom environments. We report the instruction following success rate, the percentage of total instructions that were correctly followed across all evaluations. We find that $\pi_{0.7}$ significantly outperforms $\pi_{0.5}$ and $\pi_{0.6}$ across the board, attaining high absolute success rates. 106

7.10 **Following complex referential instructions:** $\pi_{0.7}$ and prior models all succeed on the simpler re-arrangement instructions (instructions include "pick up the spoon", "put the spoon to the left of the fork" and "put the spoon to the right of the fork"), but $\pi_{0.7}$ performs significantly better on the complex and unusual instructions (instructions include "pick up the largest bowl on the table", "pick up the object I would use to eat soup" and "pick up the fruit on the largest plate"). Including subgoal images generated by a lightweight world model ($\pi_{0.7}$ (GC)) further boosts instruction following performance, making $\pi_{0.7}$ significantly more capable at following complex instructions. 107

7.11 **Breaking dataset biases by following instructions:** the improved language-following performance of $\pi_{0.7}$ enables it to break strong dataset biases. Prior models struggle on these data bias challenge tasks, which require following instructions that contradict the pattern in the data (e.g., putting dishes in the trash and trash in the dish bin). $\pi_{0.7}$, however, can follow instructions sufficiently well that it can break these strong biases to still perform the task. Notably, for the "Reverse Fridge to Microwave" task, including subgoal images from the world model in the context ($\pi_{0.7}$(GC)) is critical for success. 108

7.12 **Cross-embodiment transfer:** Left: Both $\pi_{0.7}$ and prior models achieve strong cross-embodiment transfer directly out of the box on simpler re-arrangement or repositioning style tasks. For example, for the "Table Setting" task, data was collected with a variety of different robots, and the task was tested on the static bimanual robot. All of the models performed well. For tasks that required transfer from the bimanual UR5e robot to the smaller static bimanual robot ("Bag In Backpack" and "Organize Tupperware"), the embodiment gap is larger, as there is only one source robot that is bigger and heavier than the target robot. Here, $\pi_{0.5}$ performed very poorly, but $\pi_{0.6}$ still performs quite well. Transferring from the smaller static bimanual robot to the UR5e for the "Shirt Bagging" task introduces the largest embodiment gap, and here the $\pi_{0.7}$ model significantly outperforms prior models. Right: for the more dexterous tasks that require folding towels and t-shirts, the embodiment gap poses an even greater challenge. Data for these tasks was collected with the smaller static bimanual robot, and the task was evaluated on the larger bimanual UR5e platform. $\pi_{0.7}$ was able to transfer this task successfully, and improved even more when using visual subgoal images generated by our lightweight world model. In fact, task progress matches the "zero-shot" performance of our most experienced human teleoperators, who have operated a diverse set of robots and attempted this task on the UR5e for the first time. 109

7.13 **Cross-embodiment transfer produces emergent strategies adapted to the target embodiment.** (a) On the source robot, human teleoperators use one arm to hold the bag open while the other performs insertion. On the UR5e target robot, $\pi_{0.7}$ instead discovers a single-arm pick-and-place strategy suited to the robot's greater reach. (b) Human teleoperators approach the shirt with a tilted end-effector on the source robot, while $\pi_{0.7}$ produces vertical grasps on the UR5e, which is more suitable for the larger robot's arm placement. In both cases, the policy goes beyond replicating source behavior, discovering manipulation strategies for the task that are better suited to the target embodiment. 111

7.14 **Example of language coaching:** We can "teach" a new task to $\pi_{0.7}$ by providing step-by-step verbal instructions. Because of its language following ability, $\pi_{0.7}$ can perform new tasks successfully under user instruction, and these instructions can then by used to train a high-level policy that prompts $\pi_{0.7}$ so that it can perform the task fully autonomously. 112

7.15 **Coaching to perform new long-horizon tasks:** Because $\pi_{0.7}$ can follow language instructions effectively, even for unfamiliar skills, it can be "coached" to perform a number of unseen, longer horizon tasks both when conditioned on language and generated subgoal images ($\pi_{0.7}$ (GC)). Prior models generally lack the language following ability needed to follow the coaching commands, and thus achieve very poor performance. 113

7.16 **Acquiring new autonomous capabilities with coaching:** We can use the coaching episodes collected for a number of different unseen tasks to train a high-level policy to automatically prompt $\pi_{0.7}$ in accordance with the coaching episodes. This allows us to create fully autonomous policies for these tasks ($\pi_{0.7}$ (autonomous)) that closely match the performance of the policy with live human coaching ($\pi_{0.7}$ (coaching)) *without collecting any additional data with teleoperation or any other kind of low-level actions*. 114

7.17 **Performing new short-horizon tasks:** $\pi_{0.7}$ can perform a number of new short-horizon tasks directly out of the box, including scooping rice into a rice cooker, spinning various objects such as a gear set and desk fan, and wiping down objects with a cloth, such as a ruler and headphones, despite no data being collected for any of these tasks. $\pi_{0.7}$ shows roughly equally strong performance when conditioned directly on language instructions ($\pi_{0.7}$) or generated image goals ($\pi_{0.7}$ (GC)). 115

7.18 **Scaling of generalization performance with diverse context and data:** Left: We find that $\pi_{0.7}$ (with metadata) can continuously improve its performance when it is trained on larger datasets, even when the average quality of the data actually decreases. By contrast, without training on rich conditioning information, we find that $\pi_{0.7}$ (without metadata) actually can degrade in performance as more lower quality data is introduced. Right: When $\pi_{0.7}$ is trained without our robot data with the highest task diversity, its performance degrades substantially. This suggests that $\pi_{0.7}$ is able to utilize the task diversity in our robot data to drive substantial improvements in compositional task generalization. 116

A.1 **Evaluation Tasks.** Replicated from the main text for convenience. We evaluate Octo on 9 real robot setups across 4 institutions in zero-shot and finetuning scenarios. 152

B.1 **Flow matching timestep sampling distribution**. We sample τ from a shifted beta distribution that emphasizes lower timesteps (corresponding to noisier actions), and does not sample timesteps at all above a cutoff value s. We use $s = 0.999$ in our experiments. 159

C.1 **Top left:** The graph of the value $\frac{1-\tau}{\tau \cdot r_\tau^2}$ from Eq. 4.2, which we clip at β. At $\tau = 0$, clipping is needed to make the value finite. With 5 denoising steps, if $\beta \geq 4.25$, the clipping only determines the guidance weight for the first step ($\tau = 0$). **Top right:** An ablation of β in our simulated benchmark. Increasing β provides no marginal benefit beyond $\beta = 5$. **Bottom left:** Example real robot action chunks generated from the same noise with 5 denoising steps ($n = 5$) and 100 denoising steps ($n = 100$), with lower opacities corresponding to higher guidance weight clipping ($\beta = \{5, 20, 50, 150\}$). With 5 denoising steps, the generated action chunks diverge when β is too high. **Bottom right:** β vs. maximum acceleration (second discrete difference) for a batch of 325 action chunks generated with $d = 15$ and $n = 5$. Higher β leads to more jerkiness, a proxy for out-of-distribution actions. 164

C.2 **Left:** Simulated ablation over different schedules for soft masking weights (Eq. 4.5). Exponential decay performs the best overall, although linear decay is very close behind. **Right:** Comparison with the inpainting algorithm from Diffuser [140], which overwrites a portion of the action chunk with the desired actions at each denoising step. While this simpler (and cheaper) inpainting method still provides some benefit, it is outperformed by our guidance-based approach. 165

E.1 **(Examples of extracted bounding boxes for MOO)**. Despite our best attempts, the bounding boxes extracted at train time are noisy due to the unstructured nature of BridgeData V2. This might have led to MOO's poor performance in our evaluations. 173

E.2 **(UniPi (HiP) example generations)**. Observe that the model is not able to generate a clear robot gripper, and produces videos that are blurry and visually poor. 176

E.3 **(UniPi (Ours) example generations and the resulting robot behaviors)**. While the robot succeeded in a few trials, we observed that in most cases the generated videos suffered from hallucination and physical inconsistency, which confused the low-level policy. 177

E.4 **(Example rollouts in CALVIN.)** Visualization of SuSIE-generated subgoals and trajectory rollouts on a successful 5-instruction language chain in CALVIN. 178

F.1 The $\pi_{0.7}$ model and its world model (for generating subgoal images) use several different nontrivial attention patterns during training and inference. From top left: in absence of image goals we use the same attention patterns as in $\pi_{0.5}$, with global bidirectional attention between embeddings for all memory-aware image views. Note that the FAST tokens (only available at training time) and the flow actions do not attend to each other. When image goals are present, we include them as an additional block-causal bidirectional block, after the text prompt. When we do classifier-free guidance (at inference time), we pack both the positive and negative example into the same sequence for efficient inference by constructing an "attention tree" with two branches (positive and negative), which do not attend to one another. Following BAGEL, at training time the world model receives three copies of the image, each of which is block-bidirectional within the multiview group: current observation, encoded with ViT; current observation encoded with VAE; noisy image goal encoded with VAE. The world model uses a similar inference-time CFG trick, albeit with a more complex mask because it has three CFG groups rather than two. 181

F.2 **Joint vs. end-effector control for prior models on cross-embodiment tasks.** We compare joint-space and end-effector (EE) control for baseline policies across a range of tasks, observing no substantial difference in performance between the two control modes. 182

F.3 **Operator experience in the human subject study.** Box plots show teleoperation experience (in hours) of the ten recruited operators across three categories: UR5e (target robot), the static bimanual robot (source robot), and all robots combined. The selected operators rank within the top 2% of our operator fleet in terms of teleoperation experience. 183

F.4 **Comparison of $\pi_{0.7}$ (GC) and human.** We find that $\pi_{0.7}$ (GC) achieves competitive performance compared to the human operators, in the shirt folding task with the UR5e bimanual platform. 184

List of Tables

2.1 **Finetuning Evaluation.** Octo enables data-efficient finetuning to new domains and out-performs training from scratch as well as state-of-the-art pretrained visual representations. Each domain uses $\sim$ 100 target demonstrations and the same finetuning hyperparameters. In each domain, success rates are averaged over 20 trials. $*$: New observation input (force-torque proprioception). †: New action space (joint position control). 16
2.2 **Model Ablations.** We achieve best performance when using the ViT architecture, diffusion action head, and wide training data mixture. All evaluations are performed on the WidowX setup. Success rates are averaged over 40 trials across two language-conditioned tasks and two goal-conditioned tasks. 18

6.1 **(CALVIN benchmark performance)** SuSIE is able to chain together more instructions with a higher success rate than all prior methods in the zero-shot (train A, B, C $\rightarrow$ test D) setting. We provide an example rollout in Appendix E.4. 83
6.2 **(Real-world performance)** SuSIE achieves the best success rate across the board, demonstrating both high precision as well as the ability to generalize to novel environments, objects, and language commands. 84
6.3 **(Comparison to GCBC with oracle goals)** Executing generated subgoals improves performance even compared to executing privileged ground-truth final goals. 85
6.4 **(Comparison to BridgeData-only subgoal generation model)** Cotraining the subgoal generation model on the Something-Something dataset improves performance in Scenes B and C, which are unseen in BridgeData V2. 85

A.1 Octo pretraining data mixture using datasets from the Open X-Embodiment dataset [240]. 149
A.2 Hyperparameters used during training. 150
A.3 Architecture details of Octo model variants. 150
A.4 **Model Ablations.** We achieve best performance when using the ViT architecture, diffusion action head, and wide training data mixture. All evaluations are performed on the WidowX setup. Success rates are averaged over 40 trials across two language-conditioned tasks and two goal-conditioned tasks. 153

A.5 **Zero-shot Generalization Analysis.** Using the WidowX setup, we analyze the generalization capabilities of Octo-small along several different axes. Octo performs the best on in-distribution tasks and generalizes well to novel objects and environments. However, Octo struggles to generalize to skills not seen in the WidowX embodiment. Success rates are averaged over 20 trials across two tasks. 154

B.1 Inference time of our model on an NVIDIA GeForce RTX 4090 GPU. 160

C.1 Latency measurements for various inference-time methods applied to $\pi_{0.5}$ [134]. Numbers include on-GPU neural network inference only, and are averaged over 10 inference calls after 5 warmup calls. Inference runs on an NVIDIA RTX 4090 GPU using bfloat16 precision and $n = 5$ denoising steps. BID [201] slows down inference due to sampling batches of actions, whereas RTC slows down inference due to backpropagating through each denoising step. BID (no forward contrast) refers to a version of BID without the forward contrast loss, which elides the need for a second model. BID (shared backbone) refers to a version of BID optimized specifically for the π_0 architecture, where the VLM backbone (3B parameters) is shared between the strong and weak model, so only two copies of the action expert (300M parameters) are needed. Full BID requires two copies of the entire model. 165

C.2 Breakdown of **total** inference latency by component for RTC. The image resizing component happens on the CPU of the robot computer. In the mobile manipulation case, this computer is an Intel NUC portable computer with a 12th Gen Intel i7-1260P processor. In the non-mobile case, this computer is a desktop workstation with an AMD Ryzen 9 7950X processor. In both cases, the model runs on a separate workstation with an NVIDIA RTX 4090 GPU; the robot computer and the inference workstation are both connected to the same LAN via a wired Ethernet connection, and communication happens via the WebSocket protocol. Model inference uses bfloat16 precision and $n = 5$ denoising steps. Measurements are taken from 50 inference calls during a real episode rollout, and $\pm$ one standard deviation is shown. 166

C.3 Breakdown of **model** inference latency by component for vanilla $\pi_{0.5}$ and RTC. Measurements are taken from a single profiling trace for each method, run on an RTX 4090 GPU. RTC incurs a 2.5x latency increase per denoising step. 166

C.4 Hyperparameters used for RTC (Algorithm 1). In simulation, d is held constant for each experiment, so $s_{\min}$ and b are not needed. Additional hyperparameters for the simulated experiments can be found in the code. 167

E.1 **(Goal sampling hyperparameters)** The first two hyperparameters, $k_{\min}$ and $k_{\max}$, are used for training the image-editing model. k_δ is used for training the low-level policy. All 3 of these correspond to a particular dataset. k_{test} is a test-time hyperparameter that corresponds to an evaluation setup. 170

Acknowledgments

I am forever indebted to Sergey Levine, my advisor, for all the opportunities he has given me. I started doing RL and robotics research less than a year before applying to graduate school; compared to the many other "superstar" students that I've since met at Berkeley, my undergraduate career was not particularly notable. However, Sergey saw something in me, and not only welcomed me into his lab but gave me his full attention and support (as he does for all of his students). He is a brilliant research advisor, and over the years he has continually shaped my thinking for the better, from general research taste to low-level technical details. Above all, he has a unique ability to dispel apathy and replace it with ardor; to this day, a meeting with Sergey has never failed to leave me newly excited about my research.

I have had many other mentor figures throughout my career. Rainer Mueller, my high school computer science teacher, supported me throughout my juvenile years as I grew my passion for programming and computers; I am lucky that he ran (and continues to run) such an amazing computer science department, which is not something that most high schools have. During my undergraduate years at the University of Texas at Austin, I spent time in the Texas Spacecraft Laboratory, where I got my first taste of research under the mentorship of Maruthi Akella, Siddarth Kaki, Carson Schubert, and Nihal Dhamani. I can't imagine a better environment to have done so. Later, I was advised and mentored by Scott Niekum and Caleb Chuck, both of whom selflessly took on a clueless undergrad as he first learned about RL and robotics. During my first year at UC Berkeley, I was lucky enough to have Michael Janner as one of my first close collaborators. Together, we wrote a paper that I am still immensely proud of, and I learned an incredible amount in the process.

Physical Intelligence (π) was a huge part of my life throughout the second half of my PhD. Joining the company in its infancy (as an intern) and continuing to be a part of its journey as it grows has been an enormous privilege. I would like to express my gratitude to the entire π team for being my supporters and collaborators — much of their work directly supports this dissertation. In particular, I would like to thank Quan Vuong, Brian Ichter, Karol Hausman, and Lachy Groom, not only for believing in me but for demonstrating what it is like to run a fast-growing startup — a notoriously difficult thing to do — with wisdom and kindness.

RAIL has been the core of my graduate school experience, which has exceeded my greatest expectations. Each day, I arrived at a building that felt as comfortable as my own home; I worked on problems that interested me with some of my most trusted friends at my side. I will forever miss lunches, dinners, and Blue Bottle runs with friends like Aviral, Young, Michael Janner, Dhruv, Dibya, Katie, Laura, Colin, Homer, Kyle, Cat, Paul, Oier, Karl, Manan, and Sudeep. As if that weren't enough, I have also found lifelong friends in the broader BAIR community: from ski trips and house parties with Chung Min, Grace, David, Shreyas, and Alex to 7 a.m. lifts with Ameesh to tennis with Kush and Justin. I will always

celebrate the several wins (and many more losses) of the FrisBAIRs ultimate frisbee team, and especially Jacob for his infinite patience in coaching us.

On the first day of my PhD — at EECS orientation — I met my life partner, Jess.[1] I am grateful to her for always being at my side.

Finally, I would like to thank my parents, Steve and Jing Black for their unconditional love and support. They have taught me curiosity and conscientiousness that has served we well not only in my PhD but throughout my entire life, and I owe them my unending gratitude.

[1]Technically, I noticed her, but she didn't notice me. I say it counts.

Chapter 1

Introduction

The world runs on physical labor. Right now, somewhere, a human is striking a nail with a hammer; another is picking a blueberry from a bush; another is folding a T-shirt in a bedroom. Not because they want to, but because they must — because there is nothing else that can. In spite of our centuries of industrialization, automation, and overall technological progress, there is still no such thing as a general-purpose robot. Specialized machines can assemble cars and solve Rubik's cubes, but they cannot yet manipulate the most basic of objects with the common-sense judgment of a human.

If we set aside the physical world — and confine ourselves to the artificial, adolescent domain of the digital — then the picture is different. Recent years have seen immense strides in the reasoning capabilities of artificial intelligence, and the state of the art continues to advance rapidly. AI agents have already matched or even surpassed human performance in non-physical tasks such as software engineering, cybersecurity, and mathematics. At their core, these advances have been driven by large pre-trained models: massive neural networks trained on tremendous quantities of human-generated data from the target domain (in this case, the digital). One oft-cited excuse for the lack of similar progress in robotics is the dearth of such target-domain data; there is no "Internet of robot data," as it were.

That is quickly changing. Spurred by the many successes of large pre-trained models, robotics companies have begun to collect large quantities of human-generated robot training data. The methods of collection vary: from direct teleoperation of robot arms [341, 375], to specialized handheld devices [63], to head- and wrist-mounted cameras on human workers [155]. The common factor is that the resulting data is intended for large-scale, end-to-end, neural-network-based imitation learning. The hope is that, as with digital AI agents, the combination of a large enough model and a large enough dataset will eventually enable robust, general-purpose robots with human-level performance on a wide variety of tasks.

However, scale alone is not enough. Modern AI agents would not exist without stochastic

gradient descent, the transformer, and many other innovations — big and small. Conversely, even without large models or datasets, algorithmic and architectural innovations can produce impressive results; in robotics, diffusion-based policies and action chunking enabled as-yet-unseen autonomous dexterous manipulation from only a handful of human demonstrations [375, 62].

The central question of this dissertation is: how can we best take advantage of the rapidly growing corpus of robot data to train and deploy models that are maximally proficient and general? If the recent explosion of interest in collecting robot data is the fuel, then we aim to provide the engine that will propel the field forward into a future of physical automation.

We begin with a basic instinct: following in the footsteps of language and vision, we should train a big transformer on all the data we have available (Part I). Unsurprisingly, the solution is not that simple; robotics presents many unique challenges that have not been encountered in prior research. For example, the inherent diversity of robotics applications entails a highly heterogeneous mix of data modalities — proprioceptive inputs, action outputs, camera observations, task specifications, and more — that a general-purpose robot learning system must deal with. Furthermore, scaling up immediately unveils a critical constraint imposed by the physical world: real-time control. Deploying large generalist policies requires developing new inference algorithms (Part II) to deal with compute-intensive, high-latency models. Finally, we have an obvious prerogative to take advantage of existing frontier AI systems, and their associated vast swaths of training data — e.g., Internet text, images, and videos — to bootstrap physical AI progress. But the best way to do so is not obvious, and it may not be as simple as throwing all the non-robot data into the training mix (Part III).

We address these challenges in three parts:

Part I: Training a Generalist Robot Policy

- In Chapter 2, we design a full model architecture and training system for a generalist robot policy. The architecture is designed for the particular needs of generalist robotics: it uses a large pre-trained backbone with small, swappable "heads" that can be re-initialized to adapt to new observation and action spaces; it takes both language commands and goal images as task specifications; and it utilizes techniques such as diffusion and action chunking that have been proven at smaller scales. We then train a series of models on a large, heterogeneous amalgamation of academic datasets. Much like early pre-trained models — such as BERT and GPT-2 [73, 262] — we illustrate the performance benefits of pre-training by fine-tuning the model on small amounts of narrow task-specific data for a variety of downstream tasks.
- In Chapter 3, we propose a larger, scaled-up generalist policy architecture with a vision language model (VLM) backbone. Applying lessons from Chapter 2, we move away

from an encoder-decoder architecture and instead propose a decoder-only mixture-of-transformers [187] design with two experts: a VLM expert for language and image understanding, and a smaller "action expert" that generates action chunks via flow matching [192]. Once again, we find that pre-training our policy on diverse robot data is critical for maximum performance on downstream tasks. Our scaled-up design enables high-frequency, dexterous control on multiple robots and completes difficult tasks like laundry folding.

Part II: Real-Time Control with Large Generalist Policies

- In Chapter 4, we tackle the issue of real-time control. Unlike digital agents, robot policies cannot "pause" their environment; unsurprisingly, we find that the inevitable latency introduced by large models (like the one from Chapter 3) and offboard compute can significantly impact performance. We propose an inference-time algorithm to remedy this issue, which applies out-of-the-box on to existing models without re-training. We show that our algorithm greatly improves performance and remains robust at high inference latencies, even on difficult, dexterous tasks.

- Although the algorithm from Chapter 4 works with no special training, it does so by adding a small amount of inference-time computational overhead. As explored in Part I, maximum performance on difficult tasks often requires fine-tuning anyway; therefore, in Chapter 5, we propose an extension to Chapter 4 that simulates inference latency during fine-tuning, eliding any extra inference-time compute. We find that this training-time variant is simple to implement and performs equally well.

Part III: Policy Generalization via World Models

- In Chapter 6, we find a simple way to incorporate semantic and visual reasoning from digital Internet data to greatly improve a robot policy. We do so by decomposing the policy into a high-level subgoal image generator (i.e., world model) and a low-level goal-reaching policy. The low-level policy has the relatively easy job of matching end effector poses and performing simple grasps; the high-level subgoal generator, though its job is much harder, is advantaged by initialization from a large-scale image generation model as well as co-training on non-robot video data. We find that this decomposition is favorable enough to outperform a flat policy with access to many times more neural network parameters and robot training data; our method is also able to recognize and manipulate novel objects not present in our small robot dataset.

- In Chapter 7, we present a system that is a scaled-up version of Chapter 6, with the architecture from Chapter 3 as the low-level policy. Our scaled-up subgoal generator produces multiple camera views at once, and is co-trained on a far larger corpus of both robot and non-robot data. Once again, we find that this hierarchical decomposition

helps the policy generalize to completely unseen tasks, as well as achieve zero-shot cross-embodiment transfer — i.e., perform a task on one robot embodiment using only demonstrations of that task from a different robot embodiment.

Finally, in Chapter 8, we conclude with a discussion of outstanding challenges and promising future directions. While recent progress in robot learning appears to be swift, there are still many obstacles standing in the way of true robot autonomy. As with digital AI agents, we expect the landscape of capabilities to shift rapidly as companies pour billions of dollars in robot data collection, manufacturing, and deployment; however, excellent research still has a critical role to play in reaping the full potential of these investments. We hope that the research presented in this dissertation represents a step forward into realizing a future of general-purpose robots — a future where humans perform physical labor only by choice, and never by necessity.

1.1 Publications

The following is a list of publications that support each chapter.

Chapter 2: Dibya Ghosh*, Homer Rich Walke*, Karl Pertsch*, Kevin Black*, Oier Mees*, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Quan Vuong, Ted Xiao, Pannag R. Sanketi, Dorsa Sadigh, Chelsea Finn, Sergey Levine. *Octo: An Open-Source Generalist Robot Policy*. RSS 2024 [315].

Chapter 3: Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, Ury Zhilinsky. *π_0: A Vision-Language-Action Flow Model for General Robot Control*. RSS 2025 [32].

Dissertation author contribution. *Model architecture design, training infrastructure, inference infrastructure, pre-training, figure design, and writing.*

Chapter 4: Kevin Black, Manuel Y. Galliker, Sergey Levine. *Real-Time Execution of Action Chunking Flow Policies*. NeurIPS 2025 [30].

Chapter 5: Kevin Black, Allen Z. Ren, Michael Equi, Sergey Levine. *Training-Time Action Conditioning for Efficient Real-Time Chunking*. arXiv 2025 [34].

Chapter 6: Kevin Black*, Mitsuhiko Nakamoto*, Pranav Atreya, Homer Walke, Chelsea

Finn, Aviral Kumar, Sergey Levine. *Zero-Shot Robotic Manipulation with Pretrained Image-Editing Diffusion Models.* ICLR 2024 [35].

Chapter 7: Physical Intelligence, Bo Ai, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Greg Balke, Kevin Black, George Bokinsky, Shihao Cao, Thomas Charbonnier, Vedant Choudhary, Foster Collins, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Maitrayee Dhaka, Jared DiCarlo, Danny Driess, Michael Equi, Adnan Esmail, Yunhao Fang, Chelsea Finn, Catherine Glossop, Thomas Godden, Ivan Goryachev, Lachlan Groom, Haroun Habeeb, Hunter Hancock, Karol Hausman, Gashon Hussein, Victor Hwang, Brian Ichter, Connor Jacobsen, Szymon Jakubczak, Rowan Jen, Tim Jones, Gregg Kammerer, Ben Katz, Liyiming Ke, Mairbek Khadikov, Chandra Kuchi, Marinda Lamb, Devin LeBlanc, Brendon LeCount, Sergey Levine, Xinyu Li, Adrian Li-Bell, Vladislav Lialin, Zhonglin Liang, Wallace Lim, Yao Lu, Enyu Luo, Vishnu Mano, Nandan Marwaha, Aikys Mongush, Liam Murphy, Suraj Nair, Tyler Patterson, Karl Pertsch, Allen Z. Ren, Gavin Schelske, Charvi Sharma, Baifeng Shi, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, Will Stoeckle, Jiaming Tang, Jimmy Tanner, Shalom Tekeste, Marcel Torne, Kyle Vedder, Quan Vuong, Anna Walling, Haohuan Wang, Jason Wang, XuDong Wang, Chris Whalen, Samuel Whitmore, Blake Williams, Charles Xu, Sukwon Yoo, Lili Yu, Wuming Zhang, Zhuoyang Zhang, Ury Zhilinsky. $\pi_{0.7}$*: a Steerable Generalist Robotic Foundation Model with Emergent Capabilities.* arXiv 2026 [254].

Dissertation author contribution. *Subgoal generator design, subgoal generator training, training infrastructure, inference infrastructure, figure design, and writing.*

Part I

Training a Generalist Robot Policy

Chapter 2

Octo: A Generalist Robot Policy

2.1 Introduction

The common approach for robotic learning is to train policies on datasets collected for the specific robot and task at hand. Learning from scratch in this way requires significant data collection effort for each task, and the resulting policies usually exhibit only narrow generalization. In principle, collected experience from other robots and tasks offers a possible solution, exposing models to a diverse set of robotic control problems that may improve generalization and performance on downstream tasks. However, even as general-purpose models become ubiquitous in natural language [241, 323]) and computer vision [272, 163], it has proven challenging to build the analogous "general-purpose robot model" that can control many robots for many tasks. Training a unified control policy in robotics presents unique challenges, requiring handling different robot embodiments, sensor setups, action spaces, task specifications, environments, and compute budgets.

Towards this direction, several works have proposed robotic foundation models that directly map robot observations to actions and provide zero-shot or few-shot generalization to new domains and robots. We broadly refer to these models as "generalist robot policies" (GRPs), emphasizing their ability to perform low-level visuomotor control across tasks, environments, and robotic systems [271, 37, 78, 387, 42, 286, 5, 333, 128, 351, 165]. For example, the GNM model [285] generalizes across different robotic navigation scenarios, the RoboCat model [37] handles different robot embodiments for goal-conditioned tasks, and the RT-X model [240] performs language-conditioned manipulation across five robot embodiments. Although these models represent significant steps toward a true "general-purpose robot model," they have been limited in multiple important aspects: they typically constrain downstream users to a pre-defined and often restrictive set of input observations, e.g., a single camera stream; they lack support for effective finetuning to new domains; and importantly, the largest of these models are not available to the general public.

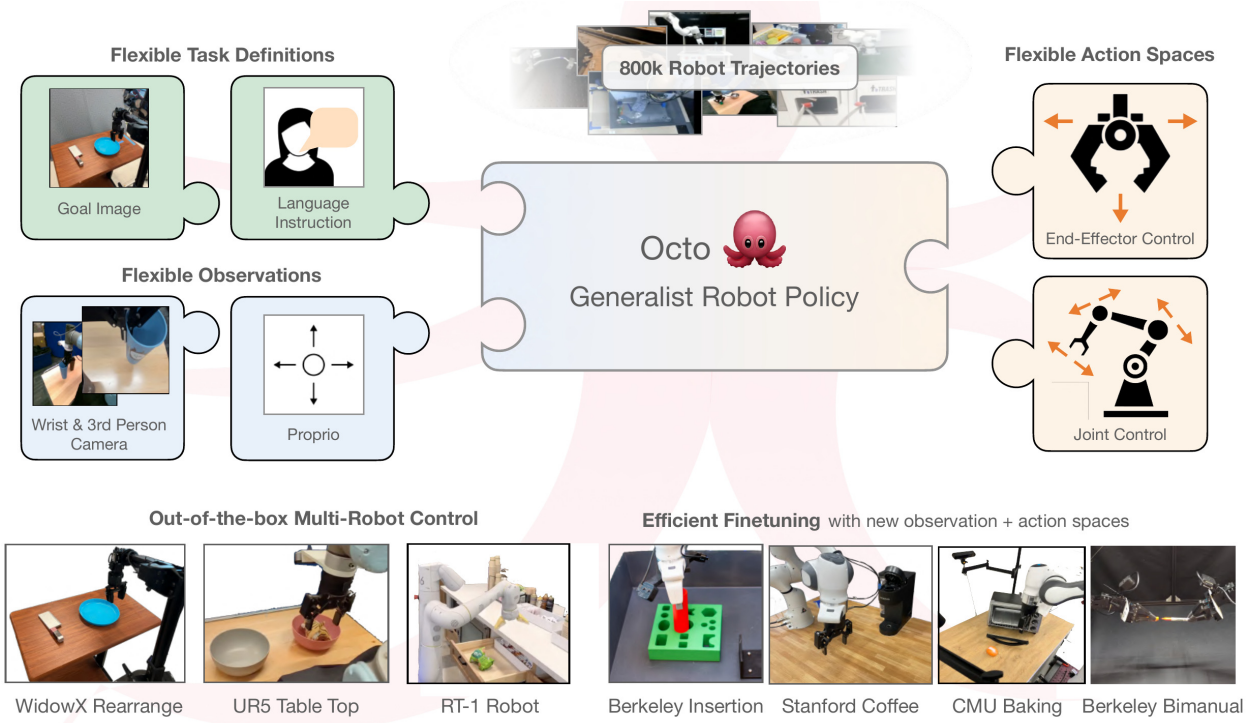


Figure 2.1: We introduce Octo, an open-source, generalist policy for robotic manipulation. Octo is a transformer-based policy pretrained on 800k diverse robot episodes from the Open X-Embodiment dataset [240]. It supports flexible task and observation definitions and can be quickly finetuned to new observation and action spaces.

In this chapter, we present a system for pretraining generalist robot policies more suitable for the diversity of interfaces in downstream robotic applications. The core of our model is a transformer architecture that maps arbitrary input tokens (created from observations and tasks) to output tokens (then decoded into actions), which can be trained on a diverse dataset of robots and tasks. With no additional training, this policy can accept different camera configurations (e.g., workspace or wrist cameras), can control different robots, and can be guided via either language commands or goal images — all by simply changing which tokens are fed into the model. Most importantly, the model can be adapted to new robot setups with new sensory inputs, action spaces, or morphologies by adding appropriate adapters and finetuning with a small target domain dataset and an accessible compute budget.

Our primary contribution is Octo, a transformer-based policy pretrained on the largest robot manipulation dataset to date: 800k robot demonstrations from the Open X-Embodiment dataset [240]. Octo is the first GRP that can be effectively finetuned to new observations and action spaces and the first generalist robot manipulation policy that is fully open-source, including the training pipeline, model checkpoints, and data. Finally, while the individual

components that comprise Octo — a transformer backbone, support for both language and goal image specification, and a diffusion head to model expressive action distributions — have been discussed in prior work, the particular combination of these components into a powerful generalist robot policy is unique and novel.

We demonstrate through extensive experiments on 9 robots across 4 institutions that our combined system leads to state-of-the-art performance for out-of-the-box multi-robot control for single and dual-arm manipulation tasks and that Octo can be used as an effective initialization for finetuning to unseen setups with new observation and action spaces. In the process, we carefully study the effect of different design decisions when pretraining GRPs; we evaluate how the choice of data distribution, model architecture, and policy formulation affects the quality of the pretrained GRP. Our evaluation highlights the utility of scale and flexibility: our best models are those trained on the widest data mixtures, with the least restrictive inductive biases, and with policy objectives that can fit the diversity of behaviors in the pretraining data.

2.2 Related Work

Many works train policies using a large dataset of trajectories collected from a robot, from early efforts using autonomous data collection for scaling policy training [255, 175, 146, 69, 94, 106] to more recent efforts that explore the combination of modern transformer-based policies with large demonstration datasets [42, 143, 375, 97, 295, 305]. These works primarily focus on a single embodiment, while Octo trains policies on robot datasets assembled across *multiple* embodiments, increasing the effective size of the training dataset and allowing finetuning to a range of robot setups.

More recently, papers have focused on broadening the generalization abilities of robot policies. Multiple works leverage diverse non-robot data or pretrained vision-language foundation models to boost policy generalization to new scenes and tasks [305, 387, 362, 58, 133, 40, 297, 130, 15, 129, 35, 14, 166, 57, 78]. More closely related to Octo are recent works that train robot policies across data from multiple robot embodiments: the GNM model [286, 285] generalizes across robot navigation setups while RoboCat [37] and RT-X [240] control multiple single-arm manipulation robots. While these models deliver impressive policy learning results, a key issue is their lack of flexibility: they typically require users to stick to the sensory inputs and action space used during pretraining and do not support adaptation to new observation and action spaces. Furthermore, the largest models are not publicly accessible. Octo differs from these works in multiple aspects: it is trained on a larger and more diverse robot data mix, it supports a wider range of downstream applications via efficient finetuning to new robot setups, and it is fully open source and reproducible.

Octo's design is inspired by several recent advances in robot imitation learning and scalable

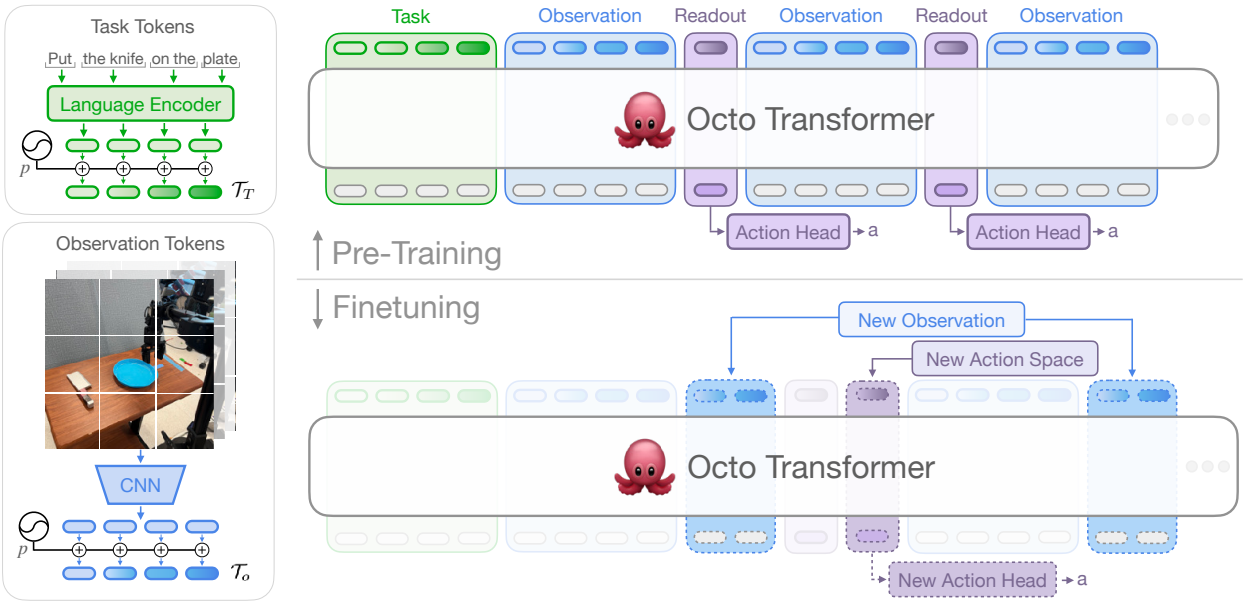


Figure 2.2: **Model architecture. Left**: Octo tokenizes task descriptions (green) and input observations (blue) using a pretrained language model and a lightweight CNN, respectively. **Top**: The transformer backbone processes the sequence of task and observation tokens and produces readout tokens (purple) that get passed to output heads to produce actions. **Bottom**: The block-wise attention structure of the transformer allows us to add or remove inputs and outputs during finetuning: for example, we can add new observations (blue, dashed) or action spaces (purple, dashed) without modifying any pretrained parameters.

transformer training, including the use of denoising diffusion objectives [120] for action decoding [61, 109, 303], the prediction of "action chunks", i.e., sequences of future actions [375, 61, 97], and model layouts and learning rate schedules inspired by the literature on scalable vision transformer training [75, 367]. Our work is the first to leverage these approaches in the context of learning cross-embodied generalist policies and we find that they can lead to substantial performance improvements. In our evaluation, we present ablations to assess the importance of these components, alongside a more comprehensive list of what we found to be (un)important in Appendix A.5.

A key ingredient for training generalist robot policies is robot training data. In contrast to vision and language data that can be scraped from the web, obtaining robot data at scale is challenging and often involves significant investments in hardware and human labor. There are multiple large robot navigation and autonomous driving datasets [100, 358, 50, 307, 285, 156, 324]. In recent years, there have also been multiple efforts for building robot *manipulation* datasets of increasing scale and diversity, either collected via scripted and autonomous policies [69, 146, 148, 48, 255, 106] or human teleoperation [219, 215, 85, 328,

136, 42, 90, 22, 274, 222, 283]. Octo is trained on the Open X-Embodiment dataset [240], an effort that pooled many of these aforementioned robot datasets. The Open-X dataset contains approximately 1.5M robot episodes, of which we curate 800k for Octo training. We note that the RT-X model [240] used a more restricted subset of 350K episodes.

2.3 The Octo Model

In this section, we describe the Octo model, our open-source generalist robot policy that can be adapted to new robots and tasks — including new sensory inputs and action spaces — via finetuning. We discuss the key design decisions, training objectives, training dataset, and infrastructure. The design of the Octo model emphasizes flexibility and scale: it supports a variety of commonly used robots, sensor configurations, and actions while providing a generic and scalable recipe that can be trained on large amounts of data. It also supports natural language instructions, goal images, observation histories, and multi-modal, chunked action prediction via diffusion decoding [61]. Furthermore, we designed Octo specifically to enable efficient finetuning to new robot setups, including robots with different action spaces and different combinations of cameras and proprioceptive information. This design was selected to make Octo a flexible and broadly applicable generalist robot policy that can be utilized for a variety of downstream robotics applications and research projects.

Architecture

At its core, Octo is a transformer-based policy π. It consists of three key parts: **input tokenizers** that transform language instructions ℓ, goals g, and observation sequences $o_1, \ldots, o_H$ into tokens $\left[\mathcal{T}_l, \mathcal{T}_g, \mathcal{T}_o\right]$ (Fig. 2.2, left); a **transformer backbone** that processes the tokens and produces embeddings $e_l, e_g, e_o = T(\mathcal{T}_l, \mathcal{T}_g, \mathcal{T}_o)$ (Fig. 2.2, top); and **readout heads** $R(e)$ that produce the desired outputs, i.e., actions a.

Task and observation tokenizers We convert task definitions (e.g., language instructions ℓ and goal images g) and observations o (e.g., wrist and third-person camera streams) into a common "tokenized" format using modality-specific tokenizers (see Fig. 2.2, left):

- **Language inputs** are tokenized, then passed through a pretrained transformer that produces a sequence of language embedding tokens. We use the `t5-base` (111M) model [267].
- **Image observations and goals** are passed through a shallow convolution stack, then split into a sequence of flattened patches [75].

We assemble the input sequence of the transformer by adding learnable position embeddings p to task and observation tokens and then arranging them sequentially $\left[\mathcal{T}_T, \mathcal{T}_{o,1}, \mathcal{T}_{o,2}, \ldots\right]$.

Transformer backbone and readout heads Once the inputs have been cast to a unified token sequence, they are processed by a transformer (see Fig. 2.2, top). This is similar to prior works that train transformer-based policies on sequences of observations and actions [342, 265]. The attention pattern of the Octo transformer is block-wise masked: observation tokens can only attend causally to tokens from the same or earlier time steps $\mathcal{T}_{o,0:t}$ as well as task tokens $\mathcal{T}_T$ (green). Tokens corresponding to non-existing observations are fully masked out (e.g., a dataset without language instructions). This modular design enables us to add and remove observations or tasks during finetuning (see below). In addition to these input token blocks, we insert learned *readout tokens* $\mathcal{T}_{R,t}$ (purple). A readout token at $\mathcal{T}_{R,t}$ attends to observation and task tokens before it in the sequence, but is not attended to by *any* observation or task token — hence, they can only passively read and process internal embeddings without influencing them. Readout tokens act similarly to the `[CLS]` token in BERT, serving as a compact vector embedding of the observation sequence thus far. A lightweight "action head" that implements the diffusion process is applied to the embeddings for the readout tokens. This action head predicts a "chunk" of several consecutive actions, similar to prior work [375, 61].

Our design allows us to flexibly add new task and observation inputs or action output heads to the model during downstream finetuning. When adding new tasks, observations, or loss functions downstream, we can wholly retain the pretrained weights for the transformer, only adding new positional embeddings, a new lightweight encoder, or the parameters of the new head as necessitated by the change in specification (see Fig. 2.2, bottom). This is in contrast to prior architectures [42, 286], where adding or removing an image input or changing the task specification would require re-initializing or re-training large components of the pre-trained model.

This flexibility is crucial to make Octo a truly "generalist" model: since we cannot cover all possible robot sensor and action configurations during pretraining, being able to adapt Octo's inputs and outputs during finetuning makes it a versatile tool for the robotics community. Prior model designs that use standard transformer backbones or fuse visual encoders with MLP output heads lock in the type and order of inputs expected by the model. In contrast, switching the observation or task for Octo *does not* require re-initializing most of the model.

Training data

We train Octo on a mixture of 25 datasets from the Open X-Embodiment Dataset [240], a diverse collection of robot learning datasets. Our training mixture includes demonstration data of a variety of tasks from several robot embodiments and scenes. These datasets are heterogeneous not just in terms of the robot type, but also in the sensors (e.g., including or not including wrist cameras) and labels (e.g., including or not including language instructions). See Fig. 2.3 and Appendix A.3 for the detailed mixture. To create our training mixture D, we curate the data by first removing all Open-X datasets that contain no image streams,

as well as those that do not use delta end-effector control. We also remove datasets that are too repetitive, have a low image resolution, or consist of excessively niche tasks. For the remaining datasets, we roughly categorize them into "more diverse" and "less diverse" datasets based on the tasks and environments, and then double the weight of the more diverse datasets during training. We also down-weight a few datasets with many repetitive episodes to avoid dominating the mixture. Finally, we zero-pad any missing camera channels and align the gripper action spaces between the datasets such that a gripper command of +1 means "the gripper is open" and 0 means "the gripper is closed." While we found the resulting training mixture to work well, future work should perform a more thorough analysis of data mixture quality for pretraining general robot policies.

Training objective

We use a conditional diffusion decoding head to predict continuous, multi-modal action distributions [120, 61]. Importantly, only one forward pass of the transformer backbone is performed per action prediction, after which the multi-step denoising process is carried out entirely within the small diffusion head. We found this policy parameterization to outperform policies trained with MSE action heads or discretized action distributions [42] in both zero-shot and finetuning evaluations. To generate an action, we sample a Gaussian noise vector $x^K \sim \mathcal{N}\big(0, I\big)$ and apply K steps of denoising with a learned denoising network $\epsilon_\theta(x^k, e, k)$ that is conditioned on the output x^k of the previous denoising step, the step index k, and the output embedding e of the transformer action readout:

$$x^{k-1} = \alpha(x^k - \gamma\epsilon_\theta(x^k, e, k) + \mathcal{N}\big(0, \sigma^2 I\big)). \tag{2.1}$$

The hyperparameters α, γ, and σ correspond to the noise schedule: we use the standard cosine schedule from [238]. We train the diffusion head using the standard DDPM objective first proposed in [120], where we add Gaussian noise to the dataset actions and train the denoising network $\epsilon_\theta(x^k, e, k)$ to reconstruct the original action. For a detailed explanation of diffusion policy training, see Chi et al. [61]. We list all hyperparameters in Appendix A.4.

We use the same diffusion training objective during finetuning and update the full model, a recipe which outperformed those that freeze subsets of the pretrained parameters. In all finetuning experiments, we employ the same recipe: given a small target domain dataset with around 100 trajectories, we finetune for 50k steps using a cosine decay learning rate decay with linear warmup.

Training Details

We trained two variants of our model: Octo-Small with a transformer backbone that mirrors the size of a ViT-S, and Octo-Base with a transformer backbone that mirrors the size of a ViT-B [75].

We use the AdamW optimizer [202] with an inverse square root decay learning rate schedule [367] with weight decay of 0.1 and gradient clipping of 1.0. The ViT-B was trained for 300k steps with a batch size of 2048 using a TPU v4-128 pod, which took 14 hours. A finetuning run of the same model on a single NVIDIA A5000 GPU with 24GB of VRAM takes approximately 5 hours and can be sped up with multi-GPU training.

We train using 2 frames of observation history; in our preliminary experiments, we found significantly diminishing gains beyond the first additional frame. We use hindsight goal relabeling [12], which selects a state uniformly from the future in the trajectory to assign as the goal image, similar to prior work [207, 328, 286, 274, 222]. We apply common image data augmentations during training, and randomly zero out the language instruction or goal image per training example to enable Octo to be conditioned on *either* language instructions *or* goal images. For datasets without language annotations, we always use goal image conditioning. This enables our model to learn control mostly from self-supervised visual observations and reduces the burden on language annotation, similar to prior work on multi-context imitation learning [207, 225, 223, 222]. For more details on the choice of hyperparameters, see Appendix A.4.

Model Checkpoints & Code

We open-source all resources required to train, finetune and run our model (see `https://octo-models.github.io`):

- **Pretrained Octo checkpoints** for Octo-Small (27M params) and Octo-Base (93M params).
- **Finetuning scripts** for Octo models, in JAX.
- **Model pretraining pipeline** for Octo pretraining on the Open X-Embodiment dataset, in JAX.
- **Standalone data loaders** for Open X-Embodiment data, compatible with JAX and PyTorch.

We provide a simple example for loading and running a pretrained Octo model in Appendix A.2.

2.4 Experiments

Our experiments provide an empirical analysis of Octo, evaluating its ability to serve as a general robotic foundation model across several axes:

1. Can Octo control multiple robot embodiments and solve language and goal tasks out of the box?
2. Do Octo weights serve as a good initialization for data-efficient finetuning to new tasks and robots, and does it improve over training from scratch and commonly used pretrained representations?
3. Which design decisions in Octo matter most for building generalist robot policies?

Evaluation setups We evaluate Octo's capabilities across a representative spectrum of 9 robot learning setups at 4 institutions (see Fig. 2.4). We test Octo's ability to control different robots out-of-the-box ("zero-shot") for language and goal image tasks using robot setups that match the pretraining data, where all robots are controlled with delta end-effector control actions and the observation spaces are RGB images. We also evaluate Octo for data-efficient finetuning to new environments and tasks, including with new observations (force-torque inputs in "Berkeley Insertion"), new action spaces (joint position control in "Berkeley Pick-Up") and new robot embodiments ("Berkeley Coke" and "Berkeley Bimanual"). Each of the finetuning setups uses ~ 100 in-domain demonstrations and finetunes in < 5 hours on a NVIDIA A5000 GPU, using the same hyperparameters across all setups (see Appendix A.4). Our evaluation tasks test Octo's ability to interact with diverse objects (e.g., "WidowX BridgeV2"), solve long-horizon tasks (e.g., "Stanford Coffee") and perform precise manipulation (e.g., "Berkeley Insertion"). For more details on each evaluation setup, see Appendix A.6.

Comparisons We compare Octo's ability to control multiple robots out-of-the-box to the best openly available generalist robot policy, **RT-1-X** [240], using the released checkpoint. Similar to Octo, RT-1-X is pretrained on the Open X-Embodiment robot dataset and aims to control multiple robots zero-shot, thus providing a natural point of comparison. We also compare the zero-shot capabilities of Octo to **RT-2-X**, a 55 billion parameter vision-language model finetuned on the Open X-Embodiment dataset to produce robot actions. The RT-1-X and RT-2-X models [240] are trained on a more restricted subset of 350K episodes (compared to 800k episodes for Octo). We further compare Octo's performance as a policy initialization for data efficient finetuning to two common approaches: (1) training on the target domain demonstrations *from scratch* and (2) using pretrained visual representations. While a number of prior works have proposed other pretraining schemes for imitation finetuning [85, 80, 90], to our knowledge no prior method provides a pretrained *policy* that has been demonstrated to finetune successfully to *new* observation and action spaces. However, pretrained visual representations such as VC-1 [211] have been used in this way, and therefore we use these methods as another point of comparison.

For finetuning, we found that training our large transformer architecture from scratch overfit quickly on the small datasets. Instead, we obtained better from-scratch results using a canonical policy architecture employed by many prior works: a ResNet visual encoder with

FiLM [252] language conditioning, combined with a small transformer action decoder trained with a diffusion objective, similar to [42, 375, 61, 208]. Our instantiation of this architecture has 28M parameters (similar to RT-1 [42]). We adopt this as our from-scratch baseline (**"ResNet+Transformer Scratch"**). We also compare to a pretrained visual representation following the procedure of Majumdar et al. [212]. A ViT-B visual encoder is initialized to the VC-1 weights [212], a state-of-the-art visual representation pretrained on 4,000 hours of ego-centric videos and ImageNet, and combined with an MLP action decoder. The full model is trained to predict expert actions using an MSE loss (**"VC-1"**).

Octo Controls Multiple Robots Out-of-the-Box

	Berkeley Insertion*	Stanford Coffee	CMU Baking	Berkeley Pick-Up†	Berkeley Coke	Berkeley Bimanual†	Average
ResNet+Transformer Scratch	10%	45%	25%	0%	20%	20%	20%
VC-1 [212]	5%	0%	30%	0%	10%	50%	15%
Octo (Ours)	**70%**	**75%**	**50%**	**60%**	**100%**	**80%**	**72%**

Table 2.1: **Finetuning Evaluation.** Octo enables data-efficient finetuning to new domains and out-performs training from scratch as well as state-of-the-art pretrained visual representations. Each domain uses ~ 100 target demonstrations and the same finetuning hyperparameters. In each domain, success rates are averaged over 20 trials. $*$: New observation input (force-torque proprioception). †: New action space (joint position control).

We compare the zero-shot manipulation capabilities of Octo, RT-1-X, and RT-2-X in Fig. 2.5. We evaluated on several tasks selected from the pre-training dataset including picking and placing, wiping a table with a cloth, and opening and closing drawers. For each robot, we selected two language tasks from the corresponding OXE dataset and performed 10 trials per task with varying initial conditions (details in Appendix A.6). The chosen tasks are "in-distribution" from the pre-training data, but the evaluation requires methods to generalize to new object positions, lighting conditions, backgrounds, and distractor objects. While all methods acted reasonably across tasks in the pretraining environments, we found that on average Octo had a 29% higher success rate than RT-1-X (35M parameters). For the WidowX and RT-1 Robot evaluations, we also compared to RT-2-X (55 billion parameters) [387] and found that Octo performed similarly.

Additionally, while RT-1-X and RT-2-X only support conditioning on language instructions, Octo also supports conditioning on goal images. We evaluated our model on the WidowX tasks using goal image conditioning and found that it achieved a 25% higher success rate than when evaluated with language conditioning. This is likely because goal images provide more information about how to achieve the task. In the BridgeV2 domain, we performed a

[01]For the WidowX, since RT-2-X is not openly available, we report the RT-2-X numbers from [35] (dashed bar) and use the same tasks for the Octo and RT-1-X evaluations. For the RT-1 Robot, the authors of RT-2-X kindly performed the evaluations for us.

fine-grained analysis of the zero-shot capabilities in Table A.5; measuring performance on setups seen in the dataset, and for novel environments, scenes, and skills. While the Octo model achieves high success on novel objects, zero-shot performance slightly degrades in a new scene, and high degradation for novel behaviors like flipping or precise insertion.

Octo Enables Data-Efficient Learning in New Domains

We report data-efficient finetuning results to new domains in Table 2.1. We find that finetuning Octo leads to better policies than starting from scratch or with the pretrained VC-1 weights. On average across the six evaluation setups (detailed in Appendix A.6), Octo outperforms the next best baseline by 52%. Importantly, we use the same recipe and hyperparameters for fine-tuning Octo on all evaluation tasks (see Section 2.3), making this a good default configuration.

The results also underline Octo's ability to accommodate new observations (force-torque inputs for "Berkeley Insertion"), action spaces (joint position control for "Berkeley Pick-Up") and new robot embodiments ("Berkeley Coke" and "Berkeley Bimanual"). This makes Octo applicable to a wide range of single and dual arm robotic manipulation problems that go beyond a single camera input and end-effector position control.

Design Decisions for Generalist Robot Policy Training

We have demonstrated the effectiveness of Octo as a zero-shot multi-robot controller and as an initialization for policy finetuning. We next analyze the effects of different design decisions on the performance of the Octo policy. Concretely, we focus on the following aspects: (1) model architecture, (2) training data, (3) training objective, and (4) model scale. Unless noted otherwise, we perform all ablations on the Octo-Small model due to our compute budget.

Model architecture Prior transformer-based policy designs typically encode input images with large ResNet-style [115] encoders and fuse the resulting image features with a comparatively small transformer [42, 240, 286, 61, 375, 223, 295]. Instead, we opt for a "transformer-first" architecture that uses very shallow CNN patch encoders and concentrates most of the parameters and FLOPS in the transformer backbone, similar to canonical vision transformer architectures [75]. In Table 2.2 we show that this scalable architecture leads to substantially improved performance when training on the full Open X-Embodiment data mix. Importantly, we found ResNet-based architectures to perform better than ViTs when training on small datasets, e.g., in our "from scratch" comparisons, underlining that large transformer policies are uniquely suited for scalable training on diverse datasets.

Training data Octo is trained on the most diverse cross-embodied robot dataset to date, a mix of 25 datasets that we manually curated from the Open X-Embodiment dataset [240] (see

		Aggregate Performance
	Octo-Small (Ours)	**83%**
DATA	RT-X dataset mix [240]	60%
	Single robot dataset (Bridge Data)	43%
POLICY	Discretized Action Prediction [240]	18%
	Continuous Action Prediction (MSE)	35%
ARCH	Resnet-50 + Transformer[240]	70%

Table 2.2: **Model Ablations.** We achieve best performance when using the ViT architecture, diffusion action head, and wide training data mixture. All evaluations are performed on the WidowX setup. Success rates are averaged over 40 trials across two language-conditioned tasks and two goal-conditioned tasks.

Section 2.3). We ablate the impact of this training mix by comparing to Octo models trained on a smaller mix of 11 datasets used in training the RT-X models [240] and a baseline trained only on data from the target robot domain. In Table 2.2 we show that the performance of Octo increases as we increase the number of training datasets. This suggests that expanding the data mix to even more datasets may further improve policy performance. We will leave this for future work, along with a more thorough investigation of best practices for data curation.

Training objective We compare Octo's diffusion decoding training objective (see Section 2.3) to common alternatives from prior work: simple MSE loss [36, 174] and cross-entropy loss on discretized actions [42, 387]. In Table 2.2 we find that Octo's diffusion training objective leads to substantially improved performance. This improvement is likely because the diffusion head can model multi-modal action distributions (unlike the MSE head) while retaining the precision of continuous actions (unlike the discrete head). Qualitatively, the policy acts more decisively than MSE-trained policies, and more precisely than those trained with discretized actions.

Model scale We compare Octo models of three different sizes following the ladder of common vision transformer models [367]: Octo-Tiny (10M), Octo-Small (27M), and Octo-Base (93M). In Figure 2.6 we show that the zero-shot performance of the policy scales

with increasing model size. We find that the Base model is more robust to initial scene configuration than the Small model, and is less prone to early grasp attempts, indicating the larger model has better visual scene perception.

2.5 Discussion

We introduced Octo, a large transformer-based policy pretrained on the largest robot manipulation dataset to date, 800k robot trajectories. We demonstrated that Octo can solve a variety of tasks out-of-the-box and showed how Octo's compositional design enables finetuning to new inputs and action spaces, making Octo a versatile initialization for a wide range of robotic control problems. Apart from the model itself, we have released our full training and finetuning code, alongside tools that make it easier to train on large robot datasets.

While Octo achieves strong performance in both zero-shot and finetuning evaluations, we find that the current model still has several short-comings, which we attribute in large parts to characteristics of the training data. First, we found that the current Octo model struggles with adequately processing wrist camera information. Often finetuning results were stronger when using only a third person camera instead of combining third person and wrist camera. Additionally, we notice a large difference between language-conditioned policy performance and goal-conditioned policy performance. In both cases, a lack of the respective modalities in the training data is the likely reason: only 27% of the data contains wrist camera information and only 56% of the pretraining data contains language annotations.

Expanding the data used to train Octo is a natural avenue of improvement. Since the Open X-Embodiment dataset is comprised of optimal robot demonstrations, the current model trains via imitation; future work may consider learning from sub-optimal or online interaction data that require alternative objectives. Further, while we trained and evaluated Octo exclusively on single and dual-arm manipulators; expanding to a wider set of robots that perform navigation or mobile manipulation would be an direction of high opportunity.

While Octo represents a step towards building generalist robot policies that work out-of-the-box on diverse robot setups, there remains work to improve the model, including better language conditioning, improved support for wrist cameras, and incorporating data beyond optimal demonstrations. We hope that Octo offers a simple launchpad for researchers and practitioners to access larger robotic datasets and leverage pretrained robotics models for efficient learning of new tasks and broad generalization.

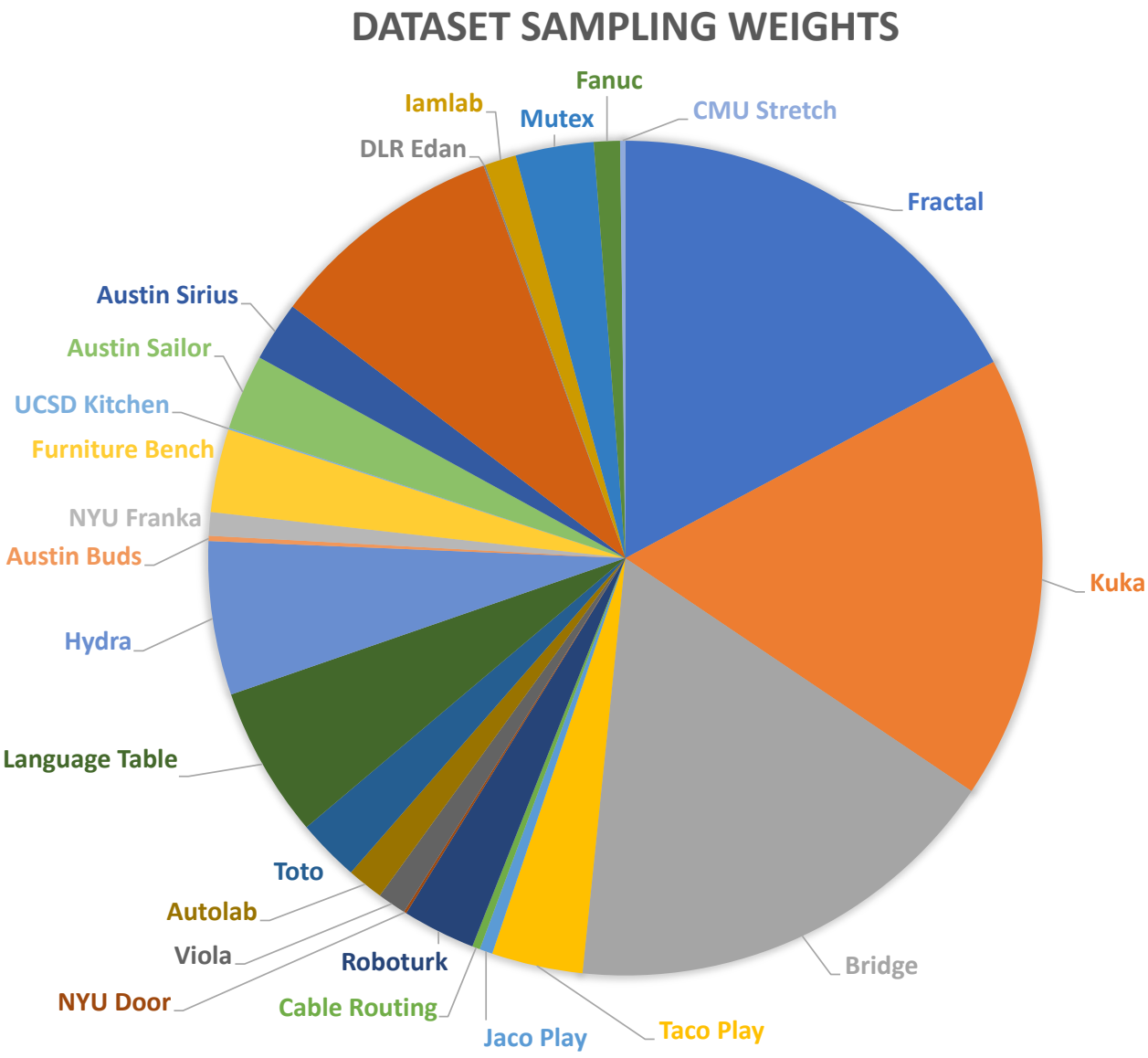


Figure 2.3: **Training dataset composition.** We curate a subset of 25 datasets from the Open X-Embodiment dataset that have image observations, end-effector actions, and show diverse behaviors. The pie chart visualizes the fractions that each dataset contributes to every training batch on average. The dataset weights are determined by the number of samples in each dataset with small modifications to balance dataset size and diversity (see Section 2.3 for details).

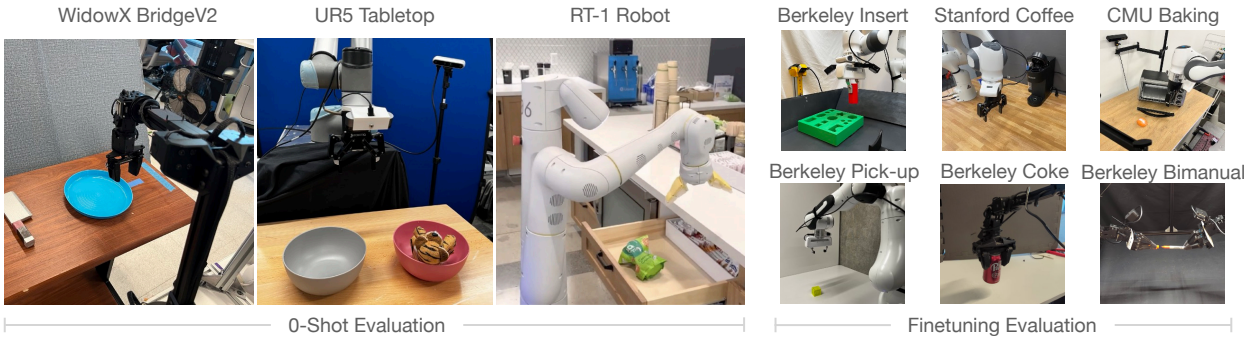


Figure 2.4: **Evaluation Tasks.** We evaluate Octo on 9 real robot setups across 4 institutions. Our evaluations capture diverse object interactions (e.g., "WidowX BridgeV2"), long task horizons (e.g., "Stanford Coffee") and precise manipulation (e.g., "Berkeley Peg Insertion"). We evaluate Octo's capabilities to control robots in environments from the pretraining data out-of-the-box and to efficiently finetune to new tasks and environments with small target domain datasets. We also test finetuning with new observations (force-torque inputs for "Berkeley Peg Insertion"), action spaces (joint position control in "Berkeley Pick-Up" and "Berkeley Bimanual") and new robot embodiments (e.g., "Berkeley Bimanual" and "Berkeley Coke").

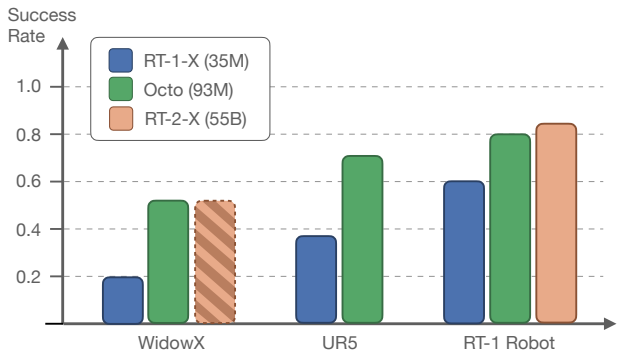


Figure 2.5: **Zero-Shot Evaluation.** Out-of-the-box, Octo can control multiple robots in environments from the pretraining data. When using natural language to specify tasks, Octo outperforms RT-1-X [240], the current best openly available generalist robot policy across three different robot embodiments and setups. Octo also performs similarly to RT-2-X [387] on the tested WidowX and RT-1 Robot tasks.[1]

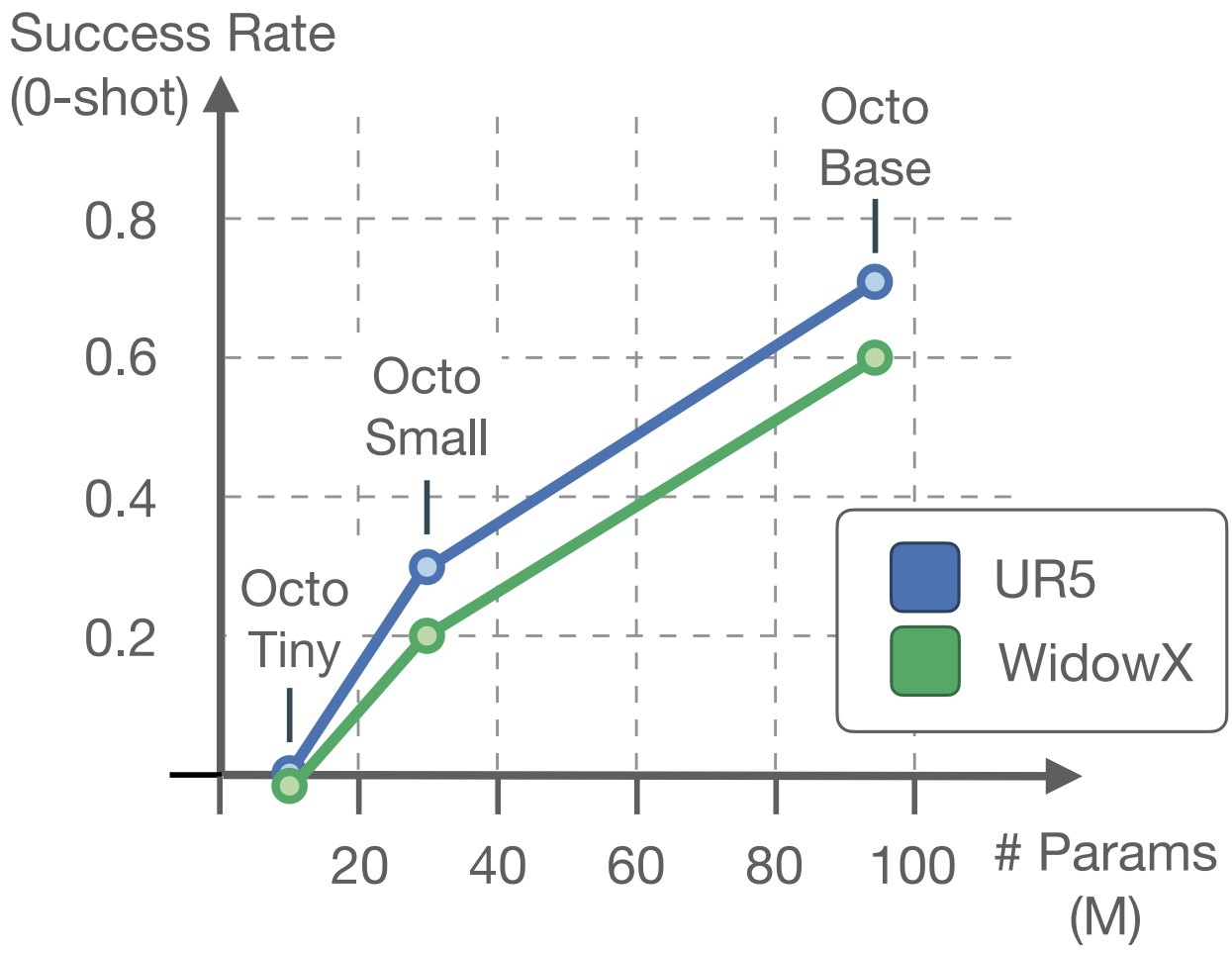


Figure 2.6: **Model Scaling.** The performance of Octo improves with larger model sizes on both UR5 and WidowX tasks. Success rates are averaged over 10 trials on one language-conditioned task per robot.

Chapter 3

π_0: A Vision-Language-Action Flow Model for General Robot Control

3.1 Introduction

Artificial intelligence systems come in all shapes and sizes, from highly specialized systems that solve complex problems inaccessible to the human mind, such as predicting the conformation of a protein [145], to systems that can produce lifelike high-resolution images or videos based on textual prompts [273]. However, the axis along which human intelligence most outpaces machine intelligence is *versatility*: the ability to solve diverse tasks situated in varied physical environments, while responding intelligently to environmental constraints, language commands, and unexpected perturbations. Perhaps the most tangible progress toward this kind of versatility in AI can be seen in large language- and vision-language models [1, 310]: systems that are pre-trained on large and very diverse corpora of images and text from the web, and then fine-tuned ("aligned") using more carefully curated datasets meant to induce the desired pattern of behavior and responsiveness. While such models have been shown to exhibit broad instruction-following and problem-solving abilities [336, 180], they are not truly *situated* in a physical world the way that people are, and their understanding of physical interaction is based entirely on abstract descriptions. If such methods are to make tangible progress toward AI systems that exhibit the kind of physically situated versatility that people possess, we will need to train them on physically situated data — that is, data from embodied robot agents.

Flexible and general-purpose models that can be tasked to perform a variety of robot behaviors have tremendous practical ramifications, but they may also offer solutions to some of the toughest challenges facing robot learning today, such as availability of data, generalization, and robustness. In natural language [1] and computer vision [263], general-purpose foundation models that are pre-trained on diverse multi-task data tend to outperform narrowly tailored

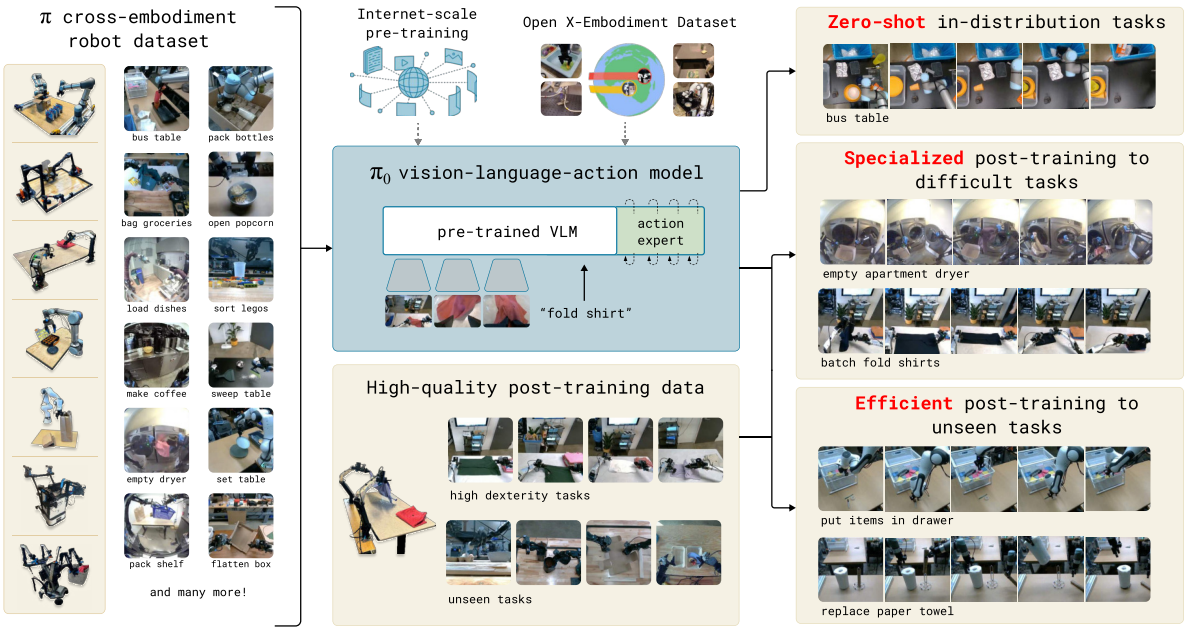


Figure 3.1: Our generalist robot policy uses a pre-trained vision-language model (VLM) backbone, as well as a diverse cross-embodiment dataset with a variety of dexterous manipulation tasks. The model is adapted to robot control by adding a separate *action expert* that produces continuous actions via flow matching, enabling precise and fluent manipulation skills. The model can then be used directly to perform tasks based on a prompt, or fine-tuned on high-quality data to enable complex multi-stage tasks, such as folding multiple articles of laundry or assembling a box.

and specialized solutions. For example, if the goal is to recognize birds in photographs, it is likely more expedient to pre-train on many different image-language associations and then fine-tune or prompt for the bird recognition task, than it is to train on only bird recognition data. Similarly, we may find that for effective specialized robot systems, it is more effective to first pre-train on highly diverse robot data, and then fine-tune or prompt for the desired task. This can resolve the data scarcity challenge, because many more sources of data are available to a generalist model — including data from other tasks, other robots, or even non-robot sources — and it may resolve robustness and generalization challenges, because the diverse data exhibits a greater coverage of observations and actions, providing a variety of scenes, corrections, and recovery behaviors that might not be present in more narrow specialized data. Thus, adopting a large-scale pre-training approach to robot learning has the potential to address many of the field's challenges and make practical learning-enabled robots a reality, while at the same time furthering our understanding of the deepest problems in artificial



Figure 3.2: **π_0 controls a mobile manipulator to fold laundry.** Our model is pre-trained on diverse data from 7 distinct robot configurations and 68 tasks, and can then either be prompted directly or fine-tuned to complex downstream tasks, as in the case of this laundry folding policy, which fetches laundry from a dryer, packs it into a hamper, brings the hamper to a folding table, and then folds each article of clothing.

intelligence.

However, developing such generalist robot policies — i.e., robot foundation models — involves a number of major challenges. First, any such research must be done at a very large scale, because the full benefits of large-scale pre-training are often not present at smaller scales [335]. Second, it requires developing the right model architectures that can effectively make use of diverse data sources, while at the same time being able to represent the intricate and subtle behaviors necessary to interact with complex physical scenes. Third, it requires the right training *recipe*. This is perhaps the most important ingredient, as much of the recent progress with large models in NLP and computer vision has relied heavily on delicate strategies for curating pre-training and post-training data [244].

In this paper, we present a prototype model and learning framework, which we call π_0, that illustrates how each of these three bottlenecks could be tackled. We illustrate our model and system in Figure 3.1. To incorporate diverse data sources, we begin by utilizing a pre-trained vision-language model (VLM) to import Internet-scale experience. By basing our model on a VLM, we inherit the general knowledge, semantic reasoning, and problem-solving abilities of language- and vision-language models. We then further train our model to incorporate robot actions, turning it into a vision-language-action (VLA) model [45]. In order to make it feasible to utilize a variety of diverse robot data sources, we employ *cross-embodiment training* [65], where data from many robot types is combined into the same model. These different robot types have different configuration spaces and action representations, including single and dual-arm systems, as well as mobile manipulators. Additionally, in order to make it possible to perform highly dexterous and intricate physical tasks, we use an action chunking architecture [375] with flow matching (a variant of diffusion) to represent complex continuous action distributions [192, 197]. This enables our model to control robots at frequencies of up to 50 Hz for dexterous tasks such as laundry folding (see Figure 3.1). To combine flow matching with VLMs, we use a novel *action expert* that augments the standard VLM with flow-based outputs.

As with language models, the architecture of our model is only part of our method. In order to flexibly and robustly perform complex tasks, we need the right training recipe. Our recipe mirrors the pre-training/post-training separation commonly seen in exascale language- and image-language models [1, 310], where the model is first pre-trained on a very large and diverse corpus, and then fine-tuned on more narrow and more carefully curated data to induce the desired pattern of behavior — in our case, dexterity, efficiency, and robustness. Intuitively, training only on high-quality data does not teach the model how to recover from mistakes, since mistakes are rarely seen in such data. Training on only lower-quality pre-training data does not teach the model to act efficiently and robustly. Combining both provides the desired behavior: the model attempts insofar as possible to act in a manner similar to the high-quality data, but still has a repertoire of recoveries and corrections that it can deploy in the case of a mistake.

The contributions of our work consist of a novel generalist robot policy architecture based on VLM pre-training and flow matching, and an empirical investigation of pre-training/post-training recipes for such robot foundation models. We evaluate our model out of the box with language commands, with fine-tuning to downstream tasks, and in combination with a high-level semantic policy that outputs intermediate language commands to perform complex and temporally extended tasks. While our model and system make use of a variety of ideas presented in recent work, the combination of ingredients is novel, and the empirical evaluation demonstrates a level of dexterity and generality that goes significantly beyond previously demonstrated robot foundation models. We evaluate our approach by pre-training on over 10,000 hours of robot data, and fine-tuning to a variety of dexterous tasks, including laundry folding (see Figure 3.2), clearing a table, putting dishes in a microwave, stacking eggs into a carton, assembling a box, and bagging groceries.

3.2 Related Work

Our work builds on recently proposed methods in large-scale robot learning, as well as multimodal language models. Our work is most closely related to recently proposed vision-language action (VLA) models, which use pre-trained VLMs that are fine-tuned for robot control [45, 161, 338]. Such models employ autoregressive discretization to represent actions in a manner analogous to text tokens. In contrast, our model employs a novel design that fine-tunes a VLM to produce actions via flow matching [197, 192], a variant of diffusion [121, 298]. This allows us to handle high-frequency action chunks [375] (up to 50 Hz) and highly dexterous tasks, which we show pose a major challenge for prior autoregressive VLAs [45]. This resembles a number of recent works on diffusion models for action generation [62, 383]. In contrast to these works, our model uses a pre-trained VLM backbone [19]. Our contribution is also fundamentally integrative, focusing on a *framework* for robot foundation models, including not only the model architecture itself but also a pre-training recipe, pre-training and post-training phases, and a range of real-world experiments.

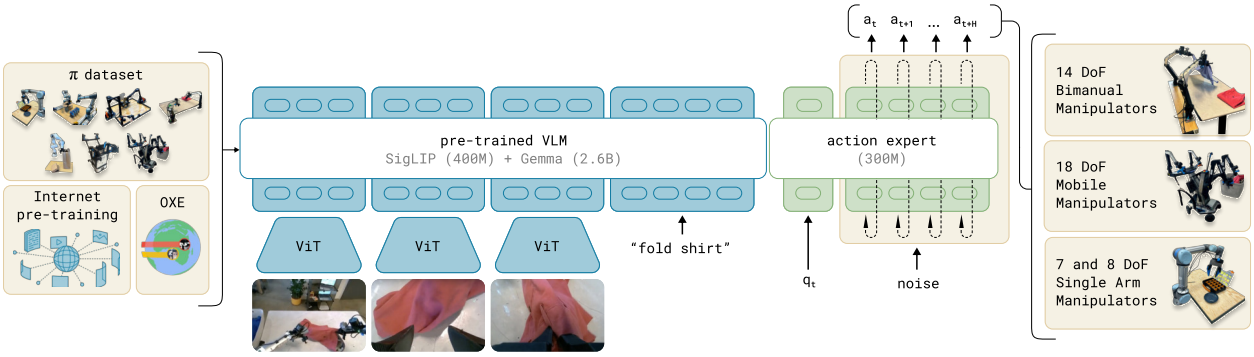


Figure 3.3: **Overview of our framework.** We start with a pre-training mixture, which consists of both our own dexterous manipulation datasets and open-source data. We use this mixture to train our flow matching VLA model, which consists of a larger VLM backbone and a smaller *action expert* for processing robot states and actions. The VLM backbone weights are initialized from PaliGemma [19], providing representations learned from large-scale Internet pre-training. The resulting π_0 model can be used to control multiple robot embodiments with differing action spaces to accomplish a wide variety of tasks.

Outside of robot control, many models have been proposed that combine pre-trained language models with diffusion [273, 275, 88], including models that specifically hybridize diffusion and autoregressive large language models [116, 193, 379]. Such models are typically concerned with image generation, but our action generation model builds on a number of previously proposed concepts. Like Zhou et al. [379], we train our model via a diffusion-style (flow matching) loss applied on individual sequence elements, in lieu of the standard cross-entropy loss for decoder-only transformers. Like Liu et al. [193], we use a separate set of weights for the tokens corresponding to diffusion. Incorporating these concepts into a VLA model, we introduce what to our knowledge is the first flow matching VLA that produces high-frequency action chunks for dexterous control.

Our work also builds on a rich history of prior works on large-scale robot learning. Early work in this area often utilized self-supervised or autonomous data collection [176, 147, 49], providing a tractable data source for simple tasks such as grasping [107, 256] or pushing [359], but without the complexity of more dexterous behaviors. More recently, a number of high-quality datasets have been collected for robot control that enable broad generalization [157, 65, 328, 218, 216, 284, 86, 21], but typically for simpler tasks that consist of object relocation and rudimentary furniture manipulation (e.g., drawer opening) [196, 89]. More dexterous tasks have been studied at a smaller scale, typically with 10s or 100s of training trajectories [375], equivalent to 10 or less hours. Since one of our aims is to study complex and dexterous behaviors, we utilize a much larger dataset, with about 10,000 hours of demonstrations, complemented by the open-source OXE dataset [65]. To our knowledge, this represents by far the largest robot learning experiment in terms of the amount of robot data. At this

scale, we show that a more sophisticated pre-training/post-training recipe is highly effective — analogously to the recipes used for large language models, a pre-training phase endows our model with a broad base of knowledge, which is then refined in a post-training phase with higher-quality curated data to achieve the desired behavior.

The complexity of the tasks we illustrate goes significantly beyond prior work. While recent work has illustrated a number of more complex and dexterous behaviors, such as tying shoelaces [373] or cooking shrimp [97], we show that our framework can learn very long tasks, sometimes tens of minutes in length, for behaviors that combine both physical dexterity and combinatorial complexity. For example, our laundry folding task requires the robot to manipulate a variety of clothing items that can start in any configuration, and fold multiple items in sequence. Our table bussing task requires discerning the class of novel objects (trash or dishes). We show that a single cross-embodiment model can be used as the base model for these tasks. To our knowledge, our work demonstrates the longest dexterous tasks in the end-to-end robot learning literature.

3.3 Overview

We provide an outline of our model and training procedure in Figure 3.3. In our training framework, we first assemble a pre-training mixture consisting of a weighted combination of our own dexterous manipulation datasets (Section 3.5), collected on 7 different robot configurations for 68 different tasks, and the entire OXE dataset [65], which contains data from 22 robots. The pre-training phase (Section 3.5) also uses diverse language labels, combining *task names* and *segment annotations* (fine-grained labels for sub-trajectories, typically about 2 seconds in length). The purpose of the pre-training phase is to train a *base model* that exhibits broad capabilities and generalization, but is not necessarily specialized for high performance on any one task. This base model can follow language commands and perform a variety of tasks at rudimentary proficiency. For complex and dexterous tasks, we then employ a post-training procedure (Section 3.5), which uses high-quality curated data to adapt the model to specific downstream tasks. We study both efficient post-training with small to moderate amounts of data, and high-quality post-training with larger datasets for complex tasks such as laundry folding and mobile manipulation.

Our model, which we describe in Section 3.4, is based on the PaliGemma vision-language model [19], which we then further train with our data mixture. To turn the base PaliGemma VLM into π_0, we add action outputs that use flow matching [197, 192] to generate continuous action distributions. We describe this design in detail in the following section. Note that we use PaliGemma for convenience and because of its comparatively small size (which is useful for real-time control), but our framework is compatible with any base pre-trained VLM.

3.4 The π_0 Model

The π_0 model, illustrated in Figure 3.3, consists primarily of a language model transformer backbone. Following the standard late fusion VLM recipe [9, 78, 194], image encoders embed the robot's image observations into the same embedding space as language tokens. We further augment this backbone with robotics-specific inputs and outputs — namely, proprioceptive state and robot actions. π_0 uses conditional flow matching [192, 197] to model the continuous distribution of actions. Flow matching provides our model with high precision and multimodal modeling capability, making it especially well suited to high-frequency dexterous tasks. Our architecture is inspired by Transfusion [379], which trains a single transformer using multiple objectives, with tokens[1] corresponding to continuous outputs supervised via a flow matching loss and tokens corresponding to discrete outputs supervised via a cross-entropy loss. Building on Transfusion, we additionally found that using a separate set of weights for the robotics-specific (action and state) tokens led to an improvement in performance. This design is analogous to a mixture of experts [290, 173, 81, 93] with two mixture elements, where the first element is used for image and text inputs, and the second is used for robotics-specific inputs and outputs. We refer to the second set of weights as the *action expert*.

Formally, we want to model the data distribution $p(\mathbf{A}_t|\mathbf{o}_t)$, where $\mathbf{A}_t = [\mathbf{a}_t, \mathbf{a}_{t+1}, ..., \mathbf{a}_{t+H-1}]$ corresponds to an *action chunk* of future actions (we use $H = 50$ for our tasks), and $\mathbf{o}_t$ is an observation. The observation consists of multiple RGB images, a language command, and the robot's proprioceptive state, such that $\mathbf{o}_t = [\mathbf{I}_t^1, ..., \mathbf{I}_t^n, \ell_t, \mathbf{q}_t]$, where $\mathbf{I}_t^i$ is i^{th} image (with 2 or 3 images per robot), ℓ_t is a sequence of language tokens, and $\mathbf{q}_t$ is a vector of joint angles. The images $\mathbf{I}_t^i$ and state $\mathbf{q}_t$ are encoded via corresponding encoders and then projected via a linear projection layer into the same embedding space as the language tokens.

For each action $\mathbf{a}_{t'}$ in the action chunk $\mathbf{A}_t$, we have a corresponding *action token* that we feed through the action expert. During training, we supervise these action tokens using a conditional flow matching loss [192, 197],

$$L^\tau(\theta) = \mathbb{E}_{p(\mathbf{A}_t|\mathbf{o}_t), q(\mathbf{A}_t^\tau|\mathbf{A}_t)} ||\mathbf{v}_\theta(\mathbf{A}_t^\tau, \mathbf{o}_t) - \mathbf{u}(\mathbf{A}_t^\tau|\mathbf{A}_t)||^2,$$

where subscripts denote robot timesteps and superscripts denote flow matching timesteps, with $\tau \in [0, 1]$. Recent work in high-resolution image [88] and video [258] synthesis has shown that flow matching can achieve strong empirical performance when combined with a simple linear-Gaussian (or optimal transport) probability path [192], given by $q(\mathbf{A}_t^\tau|\mathbf{A}_t) = \mathcal{N}(\tau\mathbf{A}_t, (1-\tau)\mathbf{I})$. In practice, the network is trained by sampling random noise $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, computing the "noisy actions" $\mathbf{A}_t^\tau = \tau\mathbf{A}_t + (1-\tau)\epsilon$, and then training the network outputs $\mathbf{v}_\theta(\mathbf{A}_t^\tau, \mathbf{o}_t)$ to match the denoising vector field $\mathbf{u}(\mathbf{A}_t^\tau|\mathbf{A}_t) = \mathbf{A}_t - \epsilon$. The action expert uses a full bidirectional

[1] In this paper, we use the word "token" to refer to an input/output slot along the sequence dimension, whether the slot corresponds to a discrete variable (e.g., a language token) or a continuous variable (e.g., an image patch or a robot action).

attention mask, so that all action tokens attend to each other. During training, we sample the flow matching timestep τ from a beta distribution that emphasizes lower (noisier) timesteps. See Appendix B.2 for more details.

At inference time, we generate actions by integrating the learned vector field from $\tau = 0$ to $\tau = 1$, starting with random noise $\mathbf{A}_t^0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. We use the forward Euler integration rule:

$$\mathbf{A}_t^{\tau+\delta} = \mathbf{A}_t^\tau + \delta \mathbf{v}_\theta(\mathbf{A}_t^\tau, \mathbf{o}_t),$$

where δ is the integration step size. We use 10 integration steps (corresponding to $\delta = 0.1$) in our experiments. Note that inference can be implemented efficiently by caching the attention keys and values for the prefix $\mathbf{o}_t$ and only recomputing the suffix corresponding to the action tokens for each integration step. We provide more details regarding the inference procedure, including the inference time for each part of the model, in Appendix B.4.

While in principle our model can be initialized from scratch or fine-tuned from any VLM backbone, in practice we use PaliGemma [19] as our base model. PaliGemma is an open-source 3 billion parameter VLM that offers a convenient tradeoff between size and performance. We add 300M parameters for the action expert (which is initialized from scratch) for a total of 3.3 billion parameters. We provide a full description of the model architecture in Appendix B.2.

Non-VLM baseline model. In addition to our main VLA model, we also trained a similar baseline model that did not use a VLM initialization for ablation experiments. This model, which we refer to as π_0-small, has 470M parameters, does not use VLM initialization, and has a number of small differences that we found to be helpful for training on our data without VLM initialization, which are summarized in Appendix B.3. This model is used in our comparisons to evaluate the benefits of incorporating VLM pertaining.

3.5 Data Collection and Training Recipe

Broadly capable robot foundation models require not only an expressive and powerful architecture, but also the right dataset and, more importantly, the right training *recipe*. In the same way that LLM training is typically divided into pre-training and post-training phases, we employ a multi-stage training procedure for our model. The goal of the pre-training phase is to expose the model to a diverse range of tasks so that it can acquire broadly applicable and general physical capabilities, while the goal of the post-training phase is to provide the model with the ability to skillfully and fluently execute the desired downstream task. Because of this, the requirements for the pre-training and post-training datasets are distinct: the pre-training dataset should cover as many tasks as possible, and within each of those tasks should cover a diversity of behaviors. The post-training dataset should instead cover behaviors that are conducive to effective task execution, which should exhibit a consistent and fluent strategy. Intuitively, the diverse (but lower quality) pre-training data allows the model

to recover from mistakes and handle highly varied situations, which might not otherwise occur in the high-quality post-training data, while the post-training data teaches the model to perform the task well.

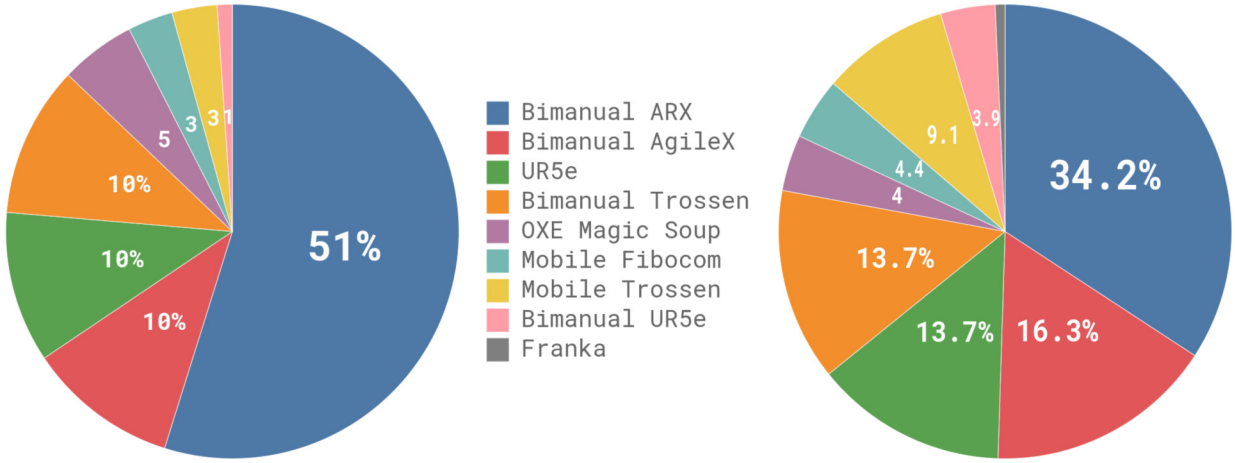


Figure 3.4: **Overview of our dataset**: The pre-training mixture consists of a subset of OXE [65] and the π dataset. We use a subset of OXE, which we refer to as OXE Magic Soup [161]. The right figure illustrates the weight of the different datasets in the pre-training mixture. The left figure illustrates their relative sizes as measured by the number of steps.

We provide an overview of our pre-training mixture in Figure 3.4. Since each training example corresponds to a timestep — i.e., a tuple $(\mathbf{o}_t, \mathbf{A}_t)$, — we will quantify data in terms of timesteps in this discussion. 9.1% of the training mixture consists of open-source datasets, including OXE [65], Bridge v2 [328], and DROID [157]. The robots and tasks in these datasets typically have one or two cameras and use low-frequency control, between 2 and 10 Hz. However, these datasets cover a wide range of objects and environments. To learn dexterous and more complex tasks, we also use 903M timesteps of data from our own datasets, where 106M steps are from single-arm robots and 797M are from dual-arm robots. This data has 68 tasks, where each task is composed of complex behaviors — e.g., the "bussing" task involves putting a wide range of different dishes, cups, and utensils into a bussing bin, and a wide array of trash items into the garbage. Note that this definition of task is significantly different from prior work, which typically uses any combination of noun and verb (e.g., "pick up the cup" vs. "pick up the plate") to constitute a distinct task. Therefore, the actual range of behaviors in our dataset is significantly broader than this number of "tasks" would imply. We discuss the specific robots and tasks in our dataset in more detail in Section 3.5.

Since the datasets are somewhat imbalanced in size (e.g., the more difficult laundry folding tasks are overrepresented), we weight each task-robot combination by $n^{0.43}$, where n is the

number of samples for that combination, such that over-represented combinations are down-weighted. The configuration vector $\mathbf{q}_t$ and action vectors $\mathbf{a}_t$ always have the dimensionality of the largest robot in the dataset (18 in our case, to accommodate two 6-DoF arms, 2 grippers, a mobile base, and a vertically actuated torso). For robots with lower-dimensional configuration and action spaces, we zero-pad the configuration and action vectors. For robots with fewer than three images, we also mask out the missing image slots.

In the post-training phase, we fine-tune our model with a smaller task-specific dataset to specialize it to particular downstream applications. As mentioned previously, our definition of "task" is fairly broad — e.g., the "bussing" task requires manipulating a wide range of different objects. Different tasks require very different datasets, with the simplest of the tasks necessitating only 5 hours and the most complex tasks using 100 or more hours of data.

Language and high-level policies

More complex tasks that require semantic reasoning and high-level strategy, such as table bussing, can also benefit from a high-level policy that decomposes high-level tasks (such as "bus the table") into more immediate subtasks (such as "pick up the napkin" or "throw the napkin into the trash"). Since our model is trained to process language inputs, we can use a high-level VLM to make these semantic inferences, a method that is analogous to LLM/VLM planning methods such as SayCan [40]. We use such a high-level policy to assist our model with high-level strategy for several of our experimental tasks, as we will discuss in Section 3.6.

Robot system details

Our dexterous manipulation datasets include 7 different robot configurations and 68 tasks. We summarize these platforms in Figure 3.5, and discuss them below:

UR5e. An arm with a parallel jaw gripper, with a wrist-mounted and over-the-shoulder camera, for a total of two camera images and a 7-dimensional configuration and action space.

Bimanual UR5e. Two UR5e setups, for a total of three camera images and a 14-dimensional configuration and action space.

Franka. The Franka setup has two cameras and an 8-dimensional configuration and action space.

Bimanual Trossen. This setup has two 6-DoF Trossen ViperX arms in a configuration based on the ALOHA setup [10, 375], with two wrist cameras and a base camera, and a 14-dimensional configuration and action space.

Bimanual ARX & bimanual AgileX. This setup uses two 6-DoF arms, and supports either

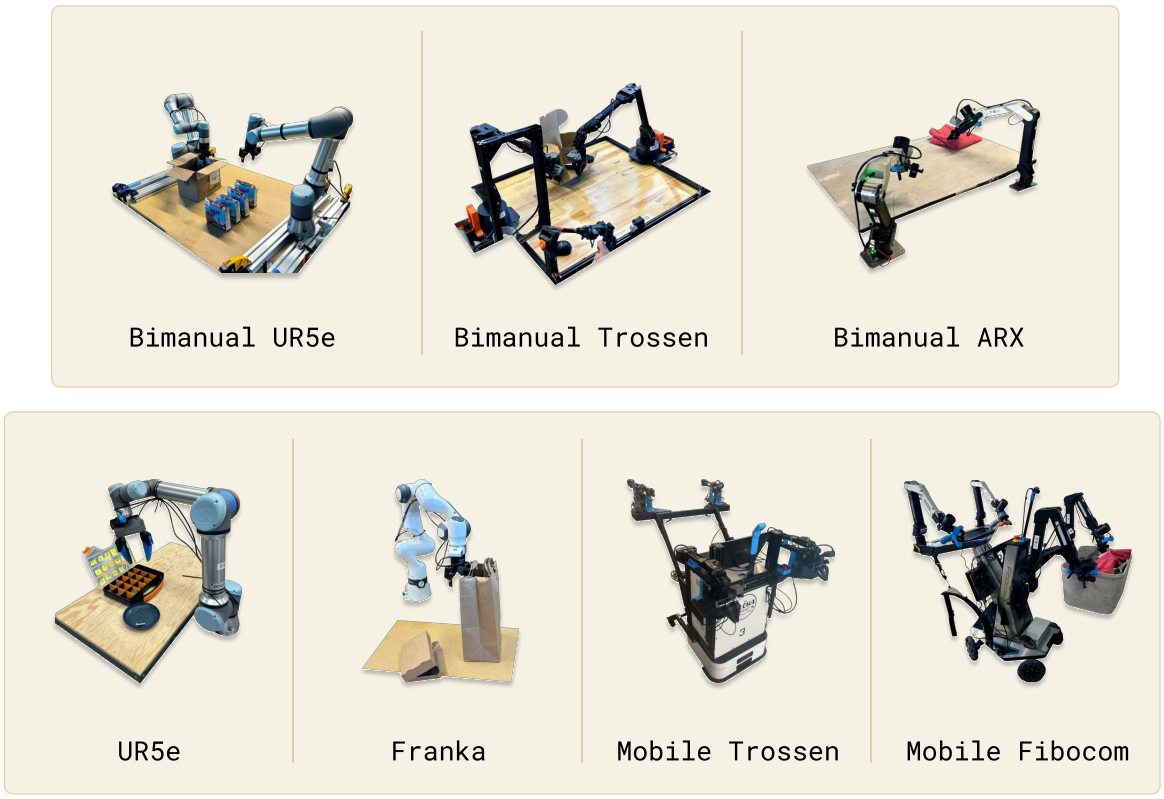


Figure 3.5: **The robots used in our experiments.** These include single and dual-arm manipulators with 6-DoF and 7-DoF arms, as well as holonomic and nonholonomic mobile manipulators. π_0 is trained jointly on all of these platforms.

ARX or AgileX arms, with three cameras (two wrist and one base) and a 14-dimensional configuration and action space. This class encompasses two distinct platforms, but we categorize them together because of their similar kinematic properties.

Mobile Trossen & mobile ARX. This setup is based on the Mobile ALOHA [375] platform, with two 6-DoF arms on a mobile base, which are either ARX arms or Trossen ViperX arms. The nonholonomic base adds two action dimensions, for a 14-dimensional configuration and 16-dimensional action space. There are two wrist cameras and a base camera. This class encompasses two distinct platforms, but we categorize them together because of their similar kinematic properties.

Mobile Fibocom. Two 6-DoF ARX arms on a holonomic base. The base adds three action dimensions (two for translation and one for orientation), for a 14-dimensional configuration and 17-dimensional action space.

We summarize the proportion of our dataset from each robot in Figure 3.4.

3.6 Experimental Evaluation

Our experimental evaluation consists of out-of-box evaluation experiments that compare our base (pre-trained) model to alternative model designs with direct prompting, as well as detailed fine-tuning experiments that evaluate our model on challenging downstream tasks, comparing it to other methods that have been proposed for dexterous manipulation. We study the following research questions:

How well does π_0 perform after *pre-training* on a variety of tasks that are present in the pre-training data? We study this question by directly evaluating π_0, with comparisons to other robot foundation models.

How well does π_0 follow language commands? These experiments compare π_0 to π_0-small, a smaller version of our model without VLM initialization, to evaluate its performance on following language commands. We evaluate with both human-provided commands and commands specified by a high-level VLM policy, as discussed in Section 3.5.

How does π_0 compare to methods that have been proposed specifically for addressing dexterous manipulation tasks? These experiments study downstream tasks for which we can either fine-tune our model from the pre-trained initialization, or train it from scratch on task-specific data, comparing to prior methods that were proposed for dexterous manipulation. We aim to evaluate both the benefits of our architecture and our pre-training procedure.

Can π_0 be adapted to complex, multi-stage tasks? In our final set of experiments, we fine-tune π_0 to a set of particularly complex tasks, including folding laundry and bussing a table. These tasks take between 5 and 20 minutes to complete. Some require guidance from a high-level policy.

Evaluating the base model

In our first set of experiments, we evaluate the model after pre-training on our full mixture, without any post-training, to evaluate how well our base model can perform a variety of tasks. We compare to other robot foundation models in the literature: both VLAs and smaller models that are trained from scratch on the same pre-training mixture. We evaluate on the following tasks, visualized in Figure 3.6, with each task commanded to the same base model via a language command.

Shirt folding: the robot must fold a t-shirt, which starts flattened.

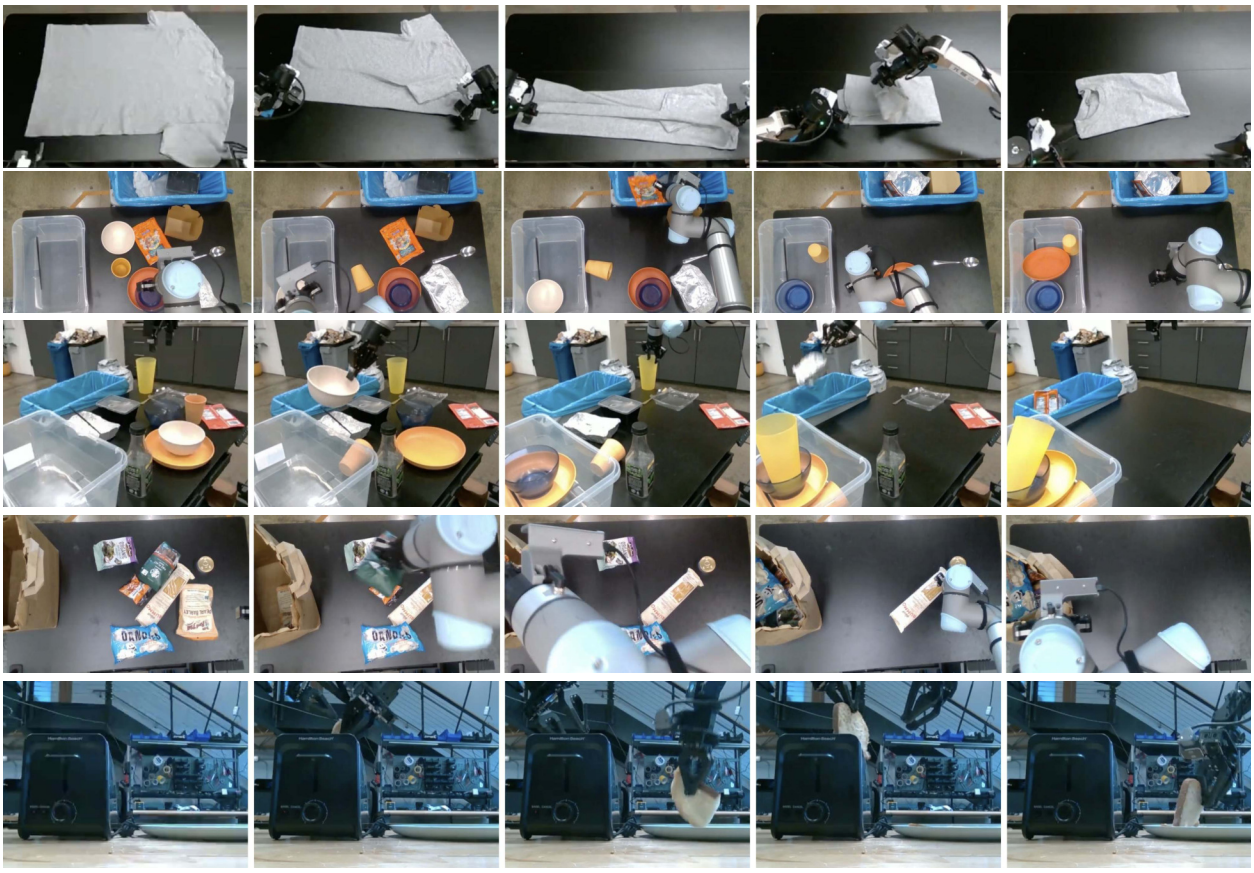


Figure 3.6: **Out-of-box evaluation tasks**: To evaluate our base model, we run it after pre-training on five tasks: **shirt folding**, **bussing easy**, **bussing hard**, **grocery bagging**, and **toast out of toaster**. The tasks require a combination of dexterous manipulation, multi-stage behaviors, and semantic recognition.

Bussing easy: the robot must clean a table, putting trash in the trash bin and dishes into the dish bin. The score indicates the number of objects that were placed in the correct receptacle.

Bussing hard: a harder version of the bussing task, with more objects and more challenging configurations, such as utensils intentionally placed on top of trash objects, objects obstructing each other, and some objects that are not in the pre-training dataset.

Grocery bagging: the robot must bag all grocery items, such as potato chips, marshmallows, and cat food.

Toast out of toaster: the robot removes toast from a toaster.

Providing comparisons for these experiments is challenging because very few prior models can operate at this scale. We compare to OpenVLA [161], a 7B parameter VLA model that was originally trained on the OXE dataset [65]. We train OpenVLA on our full mixture. This is a very difficult mixture for OpenVLA, which does not support action chunking or high-frequency control. We also compare to Octo [315], a smaller 93M parameter model. While Octo is not a VLA, it does use a diffusion process to generate actions, providing a valuable point of comparison for our flow matching VLA. We also train Octo on the same mixture as our model. Due to time constraints, we were unable to train OpenVLA and Octo for the same number of epochs as our full model. We therefore also compare to a "compute parity" version of our model, which is trained for only 160k steps (as opposed to 700k steps for our main model), which is equal to or lower than the number of steps provided to the baselines (160k for OpenVLA, 320k for Octo). We also include a version of the OpenVLA model that we fine-tuned only on the UR5e data, without cross-embodiment training, in the hopes of providing an even stronger baseline on the UR5e tasks. Finally, we include a comparison to the π_0-small model described in Section 3.4, which can be viewed as a scaled-down version of our model without VLM pre-training.

The evaluation metric uses a normalized score averaged over 10 episodes per task and method, where an episode receives a score of 1.0 for a full success, and a fractional score for partial success. For example, the score for bussing is the fraction of objects that are correctly placed in the proper receptacle. We describe the scoring rubrics in Appendix B.5. The results, shown in Figure 3.7, show that π_0 attains by far the best results across the board on all the out-of-box tasks, with near perfect success rates on shirt folding and the easier bussing tasks, and large improvements over all baselines. The "parity" version of π_0, which is trained for only 160k steps, still outperforms all the baselines, and even π_0-small outperforms OpenVLA and Octo. OpenVLA struggles on these tasks because its autoregressive discretization architecture does not support action chunks. The UR5e-only OpenVLA model performs better, but is still far below the performance of π_0. Octo does support action chunks, but has a comparatively limited representational capacity. This comparison illustrates the importance of combining large, expressive architectures with the ability to model complex distributions via flow matching or diffusion. Additionally, the comparison to π_0-small illustrates the importance of incorporating VLM pre-training. Unfortunately, it is hard to make this last comparison fair: π_0-small uses fewer parameters, but larger models are difficult to use without pre-training. Overall, these experiments show that π_0 provides a powerful pre-trained model with the ability to effectively perform a variety of tasks with a variety of robots, with much better performance than prior models.

Following language commands

In the next set of experiments, we fine-tune the base π_0 model to follow language commands in a set of evaluation domains. We compare this fine-tuned π_0 model with the π_0-small model described in Section 3.4, which we found to be the strongest baseline in the previous

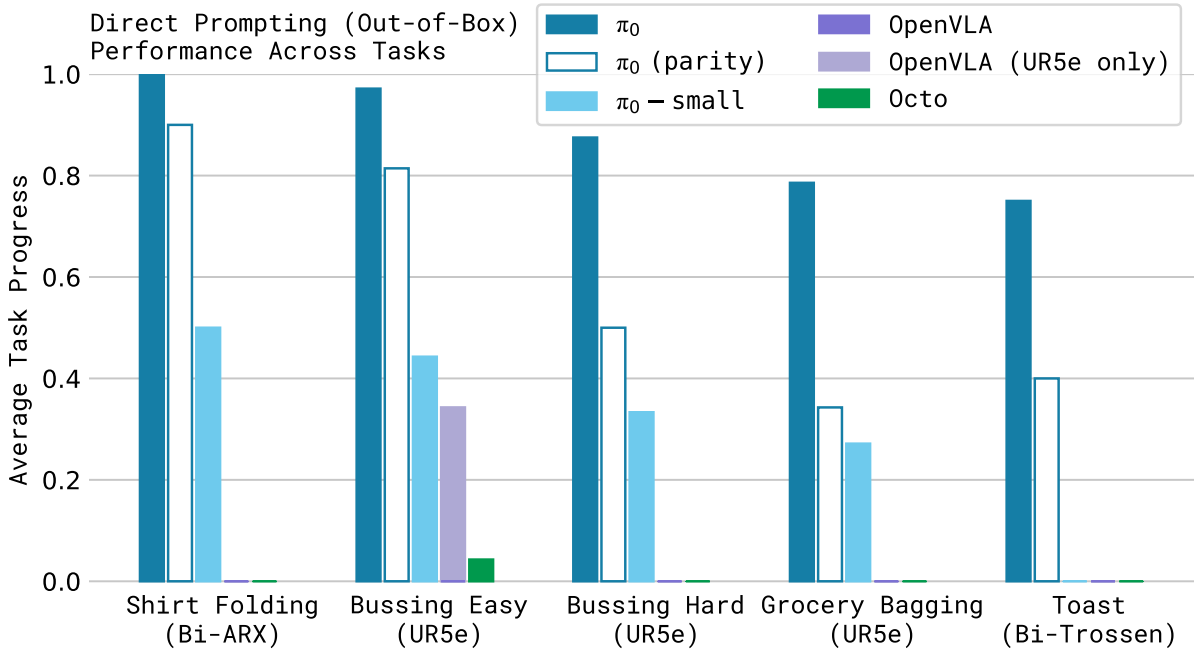


Figure 3.7: **Out-of-box evaluation results**: We evaluate π_0 trained for the full 700k steps, a version trained for 160k steps that matches the number of updates for baseline models, π_0-small, and three baselines: OpenVLA and Octo trained on all of our data, and OpenVLA trained only on the UR5e tasks (which we found to work better on UR5e tasks). Across all tasks and all comparisons, even the "parity" version of our model outperforms all baselines, and the full version of our model achieves the best results by a large margin.

section. Recall that π_0-small does *not* use a VLM initialization. This experiment therefore aims to measure how much VLM pre-training boosts our model's ability to follow language instructions. Note that π_0-small is also a significantly smaller model — unfortunately, it is difficult to remove this confounder, because VLM initialization serves both to make it practical to train a much larger model without overfitting, and to improve language instruction following. We nonetheless hope that this experiment sheds light on the language capabilities of π_0. The language instructions for each task consist of objects to pick up and locations to place those objects, with language-labeled segments that are about 2 seconds in length. Each full task consists of numerous such segments. The tasks in this evaluation consist of:

Bussing: the robot must clean a table, placing dishes and cutlery in a bin, and trash into a trash bin.

Table setting: the robot must take out items from a bin to set a table, including a place mat,



Figure 3.8: **The tasks in our language evaluation.** We evaluate our model on 3 different language-conditioned tasks, each of which requires following a sequence of intermediate language commands. The tasks involve bussing a table (top) to put dishes in a bin and garbage in a trash bin, setting a table (middle) by taking items out of a bin, and packing a shopping bag (bottom).

dishes, silverware, napkin, and cups, and adjust them according to language instructions.

Grocery bagging: the robot must pack grocery items, such as bags of coffee beans, barley, marshmallow, seaweed, almonds, spaghetti, and cans into a bag.

In Figure 3.8, we show the language-conditioned tasks in our evaluation and present the evaluation results. We evaluate five different conditions. π_0-flat (and π_0-small-flat) corresponds to directly command the model with the task description (e.g., "bag the groceries"), without intermediate language commands. π_0-human (and π_0-small-human) provides intermediate step commands (e.g., which object to pick and where to place it) from an expert human user. These conditions evaluate each model's ability to follow more detailed language commands: while these intermediate commands provide considerable information for how to perform the task, the model must be able to understand and follow those commands to benefit from them. Finally, π_0-HL evaluates π_0 with high-level commands provided by a high-level VLM, as discussed in Section 3.5. This condition is also autonomous, without any human expert.

The results in Figure 3.9, averaging over 10 trials per task, show that the language following accuracy of π_0 is significantly better than that of π_0-small. This suggests a significant improvement from the larger pre-trained VLM initialization. This capability translates to an improvement in performance with expert human guidance (π_0-human) and with high-level model guidance (π_0-HL). The results indicate that π_0's language following ability directly translates into better autonomous performance on complex tasks with high-level guidance.

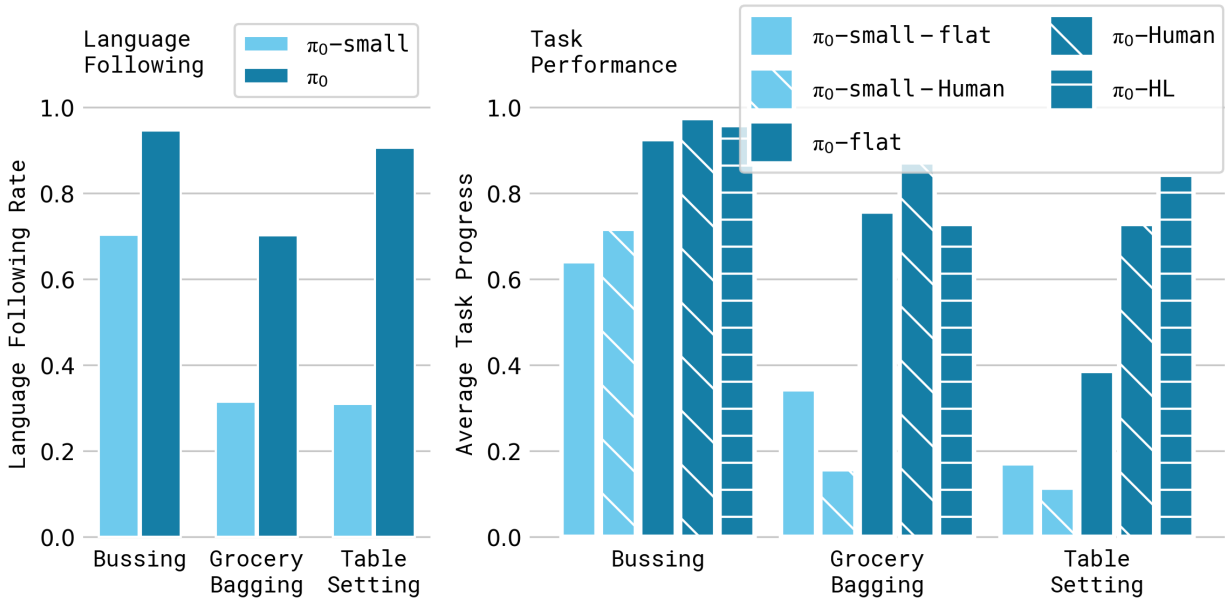


Figure 3.9: **Language evaluation.** We compare "flat" versions of our policies, −`flat`, which receive only the overall task command (e.g., "bag the groceries") with a method that receives intermediate commands from a human expert, −`human`, or a high-level VLM policy, −`HL`. We also compare our model to a small non-VLM variant under the "expert" condition, π_0 and π_0-small, in terms of language following accuracy. The results show a significant improvement with π_0 from intermediate language commands provided by a human expert and to a lesser degree by an autonomous high-level policy. Notably, due to π_0-small's limited language following ability, overall it does not gain with the addition of a high-level expert.

Learning new dexterous tasks

In the next set of experiments, we evaluate our model on new tasks that differ significantly from the pre-training data, requiring entirely new behaviors. For these evaluations, we fine-tune the model using various amounts of data for each new task. While each task is new, we partition the tasks into "tiers" depending on how much they differ from tasks in the pre-training data. The tasks, shown in Figure 3.10, are:

UR5e stack bowls. This task requires stacking bowls, with four bowls of different sizes. Since this task requires grasping and moving dishes like the bussing task in the pre-training data, we place it in the "easy" tier. The training data contains a variety of bowls, and the evaluations use a mix of seen and unseen bowls.

Towel folding. This task requires folding a towel. Since this is similar to shirt folding, which is present in pre-training, we place it in the "easy" tier.

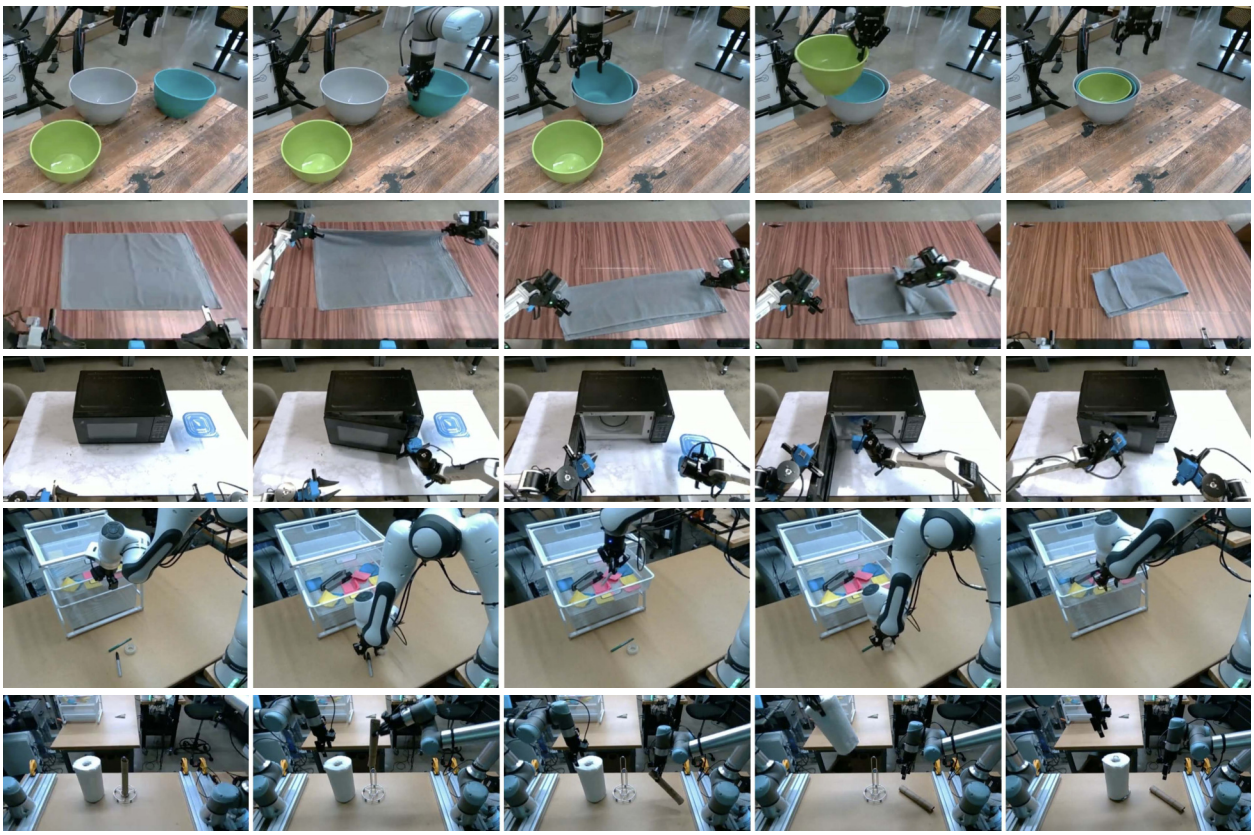


Figure 3.10: **Fine-tuning evaluation tasks**: We fine-tune our model to a variety of downstream tasks that are distinct from the tasks seen in pre-training. Our tasks represent a range of similarity from the pre-training tasks, with tasks that are most similar to pre-training (**stack bowls** and **towel folding**), a task that introduces an unseen new element (a microwave), and tasks that require new motions and new object types (**Franka items in drawer** and **paper towel replacement**).

Tupperware in microwave. This task requires opening a microwave, putting a plastic container inside it, and closing it. The containers come in different shapes and colors, and the evaluations use a mix of seen and unseen containers. The container manipulation resembles pre-training data, but the microwave is not found in pre-training.

Paper towel replacement. This task requires removing an old cardboard paper towel tube from a holder and replacing it with a fresh paper towel roll. Because no such items are found in pre-training, we consider this "hard."

Franka items in drawer. This task requires opening a drawer, packing items into a drawer, and closing it. Because there is no similar task with the Franka robot in pre-training, we

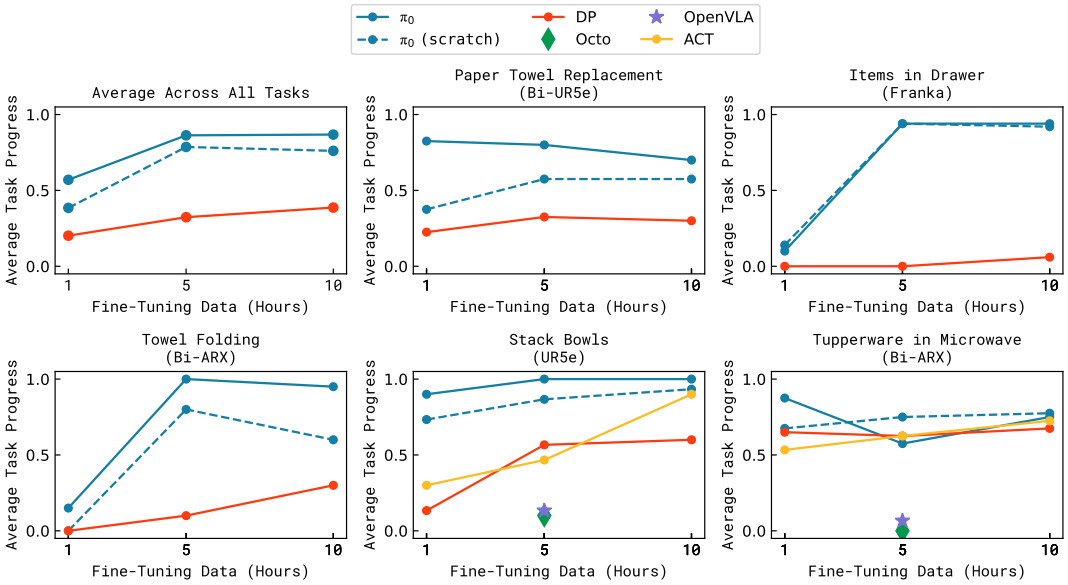


Figure 3.11: **Fine-tuning with varying amounts of data.** π_0 can learn some easier tasks even with smaller amounts of data, and the pre-trained model often attains a larger improvement over the model trained from scratch.

consider this "hard."

We compare our model after fine-tuning both to OpenVLA [161] and Octo [315], which also employ a pre-training and fine-tuning recipe. Since our aim is to evaluate the specific models (rather than the architectures), we use the publicly available pre-trained checkpoints for these models, which are trained on OXE [65], and then fine-tune them to each task. We also compare to ACT [375] and Diffusion Policy [62], which are designed specifically for learning dexterous tasks from smaller datasets. ACT and Diffusion Policy are trained *only* on the fine-tuning datasets, which are of similar size to the individual datasets used in the ACT and Diffusion Policy experiments [62, 375]. We evaluate π_0 by fine-tuning from our pre-trained base model, as well as by training from scratch. This comparison is meant to evaluate the individual benefits of the π_0 architecture and our pre-training procedure. We hypothesize that the π_0 architecture with VLM initialization should already provide a stronger starting point for the individual tasks, while the pre-training procedure should further improve its performance, especially with smaller fine-tuning datasets.

Figure 3.11 shows the performance across all of the tasks for a variety of methods, averaging over 10 trials per task, with different amounts of fine-tuning data on each task. We include all of the baselines on the **stack bowls** and **Tupperware in microwave** tasks. Since OpenVLA

and Octo attain significantly worse performance, we only run these for one of the dataset sizes, due to the time cost of evaluating so many models in the real world. The results show that π_0 generally outperforms other methods. Interestingly, the strongest prior models are the ones that are trained entirely from scratch on the target tasks, suggesting that leveraging pre-training in these domains presents a major challenge for prior approaches. While the 5-hour policy for π_0 on the Tupperware task performs similarly to the baselines, the 1-hour version is significantly better. As expected, pre-training leads to larger improvement for tasks that are more similar to the pre-training data, though the pre-trained model is frequently better than the non-pre-trained model, sometimes by as much as 2x.

Mastering complex multi-stage tasks

In our final set of experiments, we tackle a range of challenging multi-stage tasks via a combination of fine-tuning and language. For some of these tasks, data is present in pre-training, but fine-tuning is required to attain mastery. For some, no data is present in pre-training. The tasks in this evaluation, shown in Figure 3.12, are:

Laundry folding: This task requires a static (non-mobile) bimanual system to fold articles of clothing. The clothing items start in a randomized crumpled state in a bin, and the goal is to take out the item, fold it, and place it on top of a stack of previously folded items. The randomized initial configuration of the crumpled laundry presents a major challenge, since the policy needs to generalize to any configuration. This task is present in pre-training.

Mobile laundry: Here, the Fibocom mobile robot in Figure 3.5 has to fold laundry, facing many of the same challenges while controlling orientation and translation. This task is present in pre-training.

Dryer unloading: Here, the Fibocom mobile robot has to take laundry out of a dryer and place it into a hamper. This task is present in pre-training.

Table bussing: This task requires bussing a table with a diverse array of novel objects in a clutter scene, presenting a much greater challenge than the benchmark in our out-of-box evaluation: the policy must generalize to unseen objects of varying shapes and sizes, and perform complex dexterous motions, such as twisting the gripper to pick up large plates and carefully grasping thin, delicate items such as glasses. The robot must handle dense clutter and intelligently sequence various behaviors — for example, to clean off a plate with trash, it must first pick up the plate, then shake its contents into the garbage, and then place the plate in the bin. This task is not present in pre-training.

Box building: The robot has to assemble a cardboard box that starts in a flattened state. This task presents a number of major challenges: the box needs to bent in the right way, and the robot needs to hold down parts of the box while folding others, utilizing both arms

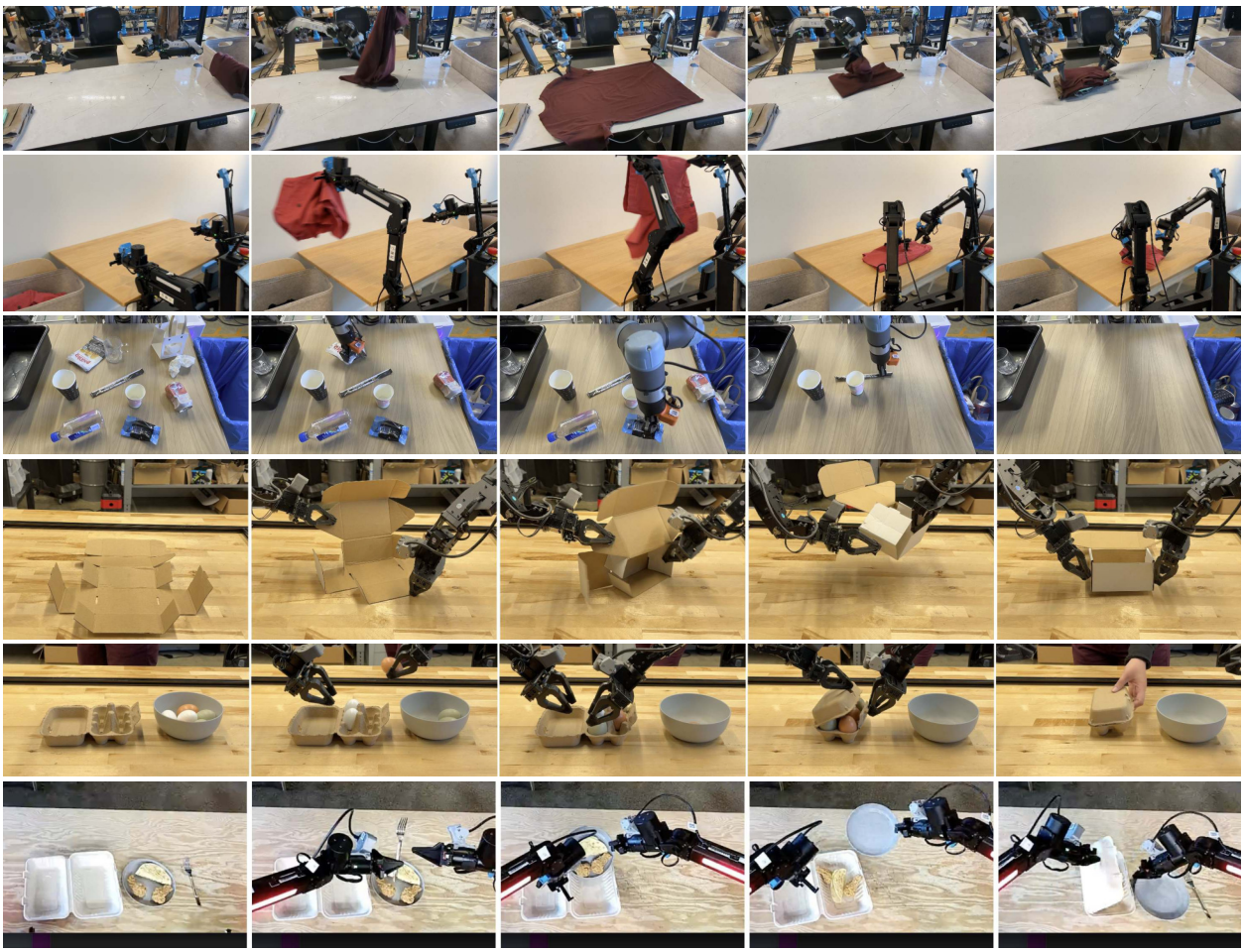


Figure 3.12: **We evaluate a range of complex and temporally extended tasks.** This includes: folding laundry from a bin with a stationary (a) or mobile (b) robot, bussing a real lunch table (c), assembling a box (d), packing eggs into a carton (e), and packing food into a to-go box (f). These tasks require combining dozens of individual behaviors, such as grasping, stacking, folding, and flattening, generalization to a huge variety of object configurations, and complex physical properties, such as deformable objects or flexible cardboard.

and even the surface of the table to brace during folding motions. The robot might need to retry some folds, requiring a reactive and intelligent strategy. This task is not present in pre-training.

To-go box: This task requires moving several food items from a plate into a to-go box, requiring packing the items into the box so that they do not stick out, and then closing the box with both arms. This task is not present in pre-training.

Packing eggs: The robot needs to take six eggs out of a bowl and pack them into an egg carton, and then close the carton. The eggs need to be grasped in a manner appropriate to their pose inside the bowl, and then placed into open slots in the carton. This presents challenges due to the egg shape, slipperiness, and the need for careful placement. Closing the box requires the use of both arms. This task is not present in pre-training.

The results, showing average scores per task over 10 trials, are presented in Figure 3.13. The scoring rubrics are in Appendix B.5. A score of 1.0 represents a perfect execution, while partial scores correspond to partially completed tasks (e.g., 0.5 indicates that half the objects were bussed correctly). These tasks are very difficult, and we were not able to solve them with other methods. We therefore use these tasks to compare to ablations of our approach, evaluating π_0 after pre-training and fine-tuning, out of the box after pre-training only ("out-of-box"), and training on the fine-tuning data without any pre-training ("scratch"). The results show that π_0 can solve many of these tasks, with our full pre-training and fine-tuning recipe performing best across the board. Note that many of these more difficult tasks show a very large improvement from using the pre-trained model, indicating that pre-training is especially useful with harder tasks. The absolute performance of π_0 varies across the tasks, likely due to differences in task difficulty and the degree to which the tasks are represented in pre-training. We recommend that readers watch the task videos on the accompanying website for a more complete impression of these tasks and their complexity. We believe that this level of autonomous performance on such challenging tasks represents a new state of the art in dexterous robot manipulation with learned policies.

3.7 Discussion, Limitations, and Future Work

We presented a framework for training a robot foundation model, which we refer to as π_0, that consists of pre-training on highly diverse data, followed by either out-of-box evaluation or fine-tuning to complex downstream tasks. Our empirical evaluation studies tasks that combine dexterity, generalization, and temporally extended multi-stage behaviors. Our model incorporates Internet-scale vision-language model (VLM) pre-training with flow matching for representing complex high-frequency action chunks. Our pre-training mixture consists of 10,000 hours of dexterous manipulation data from 7 different robot configurations and 68 tasks, in addition to large amounts of previously collected robot manipulation data from OXE [65], DROID [157], and Bridge [328]. To our knowledge, this represents the largest pre-training mixture ever used for a robot manipulation model. Our fine-tuning experiments include over 20 tasks, where we show that our model outperforms a variety of baselines, including prior VLA models [161] and models designed specifically for dexterous manipulation [375, 62]. We also examine how our post-training recipe can enable highly complex tasks, such as folding multiple articles of clothing from arbitrary initial configurations or assembling boxes.

Our framework broadly resembles the training procedures employed for large language models,

which typically consist of pre-training a base model on very large datasets scraped from the web, followed by a post-training procedure that aims to "align" the model to enable it to follow instructions and perform user commands. It is generally recognized that most of the "knowledge" in such models is acquired in the pre-training phase, while the post-training phase serves to tell the model how it should leverage that knowledge to fulfill user commands. Our experiments imply that an analogous phenomenon might take place with robot foundation models, where pre-trained models have some zero-shot capabilities, but complex tasks like laundry following require fine-tuning with high-quality data. Training on only this high-quality data results in a brittle model that does not reliably recover from mistakes, while running the pre-trained model in zero shot does not always exhibit the fluent strategies demonstrated in the post-training data.

We hope that our results will serve as a stepping stone toward general and broadly applicable robot foundation models. Our experiments suggest that such models may soon be a reality, but there are a number of limitations and ample room for future work. First, our experiments do not yet provide a comprehensive understanding of how the pre-training datasets should be composed: we combined all data available to us, but understanding what type of data is more helpful to add and how it should be weighted remains an open problem. Not all tasks in our evaluation work reliably, and it remains unclear how to predict how much and what kind of data is needed to attain near-perfect performance. Finally, it remains to be seen how much positive transfer there is in combining highly diverse data, particularly from different tasks and different robots: although our results suggest that universal pre-trained robot foundation models might become a reality, it is left for future work to understand whether this universality extends to much more distinct domains, such as autonomous driving, navigation, and legged locomotion.

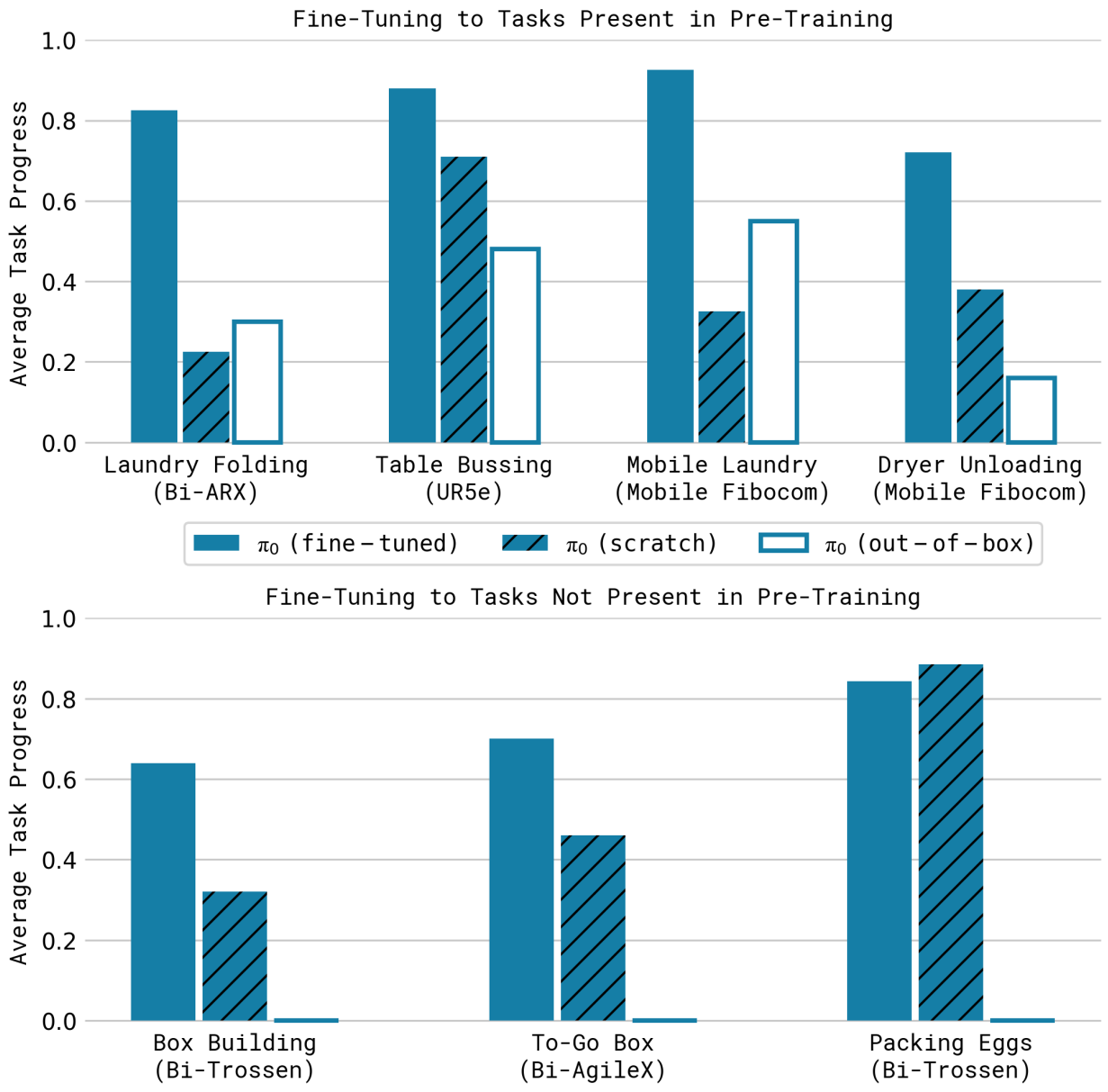


Figure 3.13: **Post-training results on complex tasks** in terms of average scores over 10 trials. The full pre-trained π_0 model attains more than 50% of the maximum score across all of the tasks, and typically outperforms the ablations, with especially significant improvements on the hardest tasks.

Part II

Real-Time Control with Large Generalist Policies

Chapter 4

Real-Time Chunking

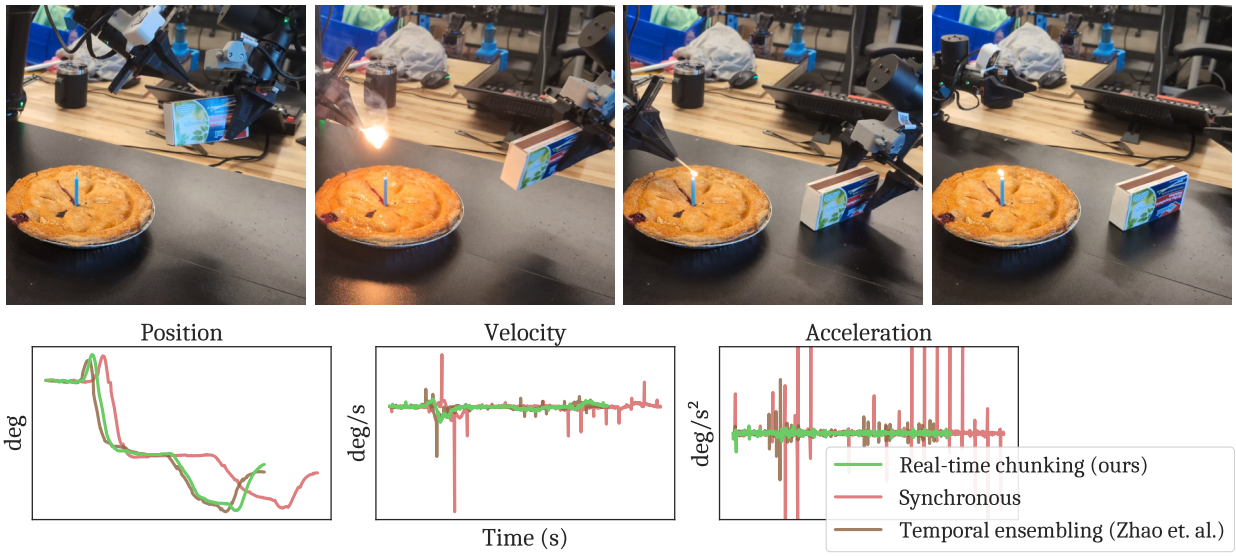


Figure 4.1: **Top:** Real-time chunking (RTC) enables the robot to perform highly dexterous and dynamic tasks, such as lighting a match—even in the presence of inference delays in excess of 300 milliseconds, corresponding to more than 30% of the model's prediction horizon. **Bottom:** RTC performs the same robot motion 20% faster than synchronous inference [32, 161, 44, 134, 159, 315], and smoother than all competing methods, including temporal ensembling [375]. The shown positions, velocities, and accelerations correspond to the shoulder joint of one arm, and are taken from the first 10 seconds of a real autonomous match-lighting rollout.

4.1 Introduction

As AI systems have become more capable, they have also interacted more and more directly with their environment. Whether they're executing terminal commands [242], playing Pokémon on livestream [108], or browsing the web on your behalf [334], recent advances—driven primarily by large-scale deep learning—have enabled these systems to increasingly *control*, rather than merely *process*, the vast heterogeneity of the outside world. Embodied agents, where machine learning models directly control real, physical constructs, are perhaps the quintessential example. The same advances fueling agentic language and vision models are also making great strides in physical intelligence on platforms ranging from humanoid robots [28] to autonomous cars [320].

Cyber-physical systems, unlike chatbots and image generators, always operate in *real time*. While a robot is "thinking", the world around it evolves according to physical laws. Thus, delays between inputs and outputs have a tangible impact on performance. For a language model, the difference between fast and slow generation is a satisfied or annoyed user; for a robot action model, on the other hand, it could be the difference between a robot handing you a hot coffee or spilling it in your lap.

Unfortunately, the effectiveness of modern large-scale machine learning comes with high latency as an unavoidable side effect. Large language models (LLMs), vision-language models (VLMs), and vision-language-action models (VLAs)—the last referring to a class of models designed for visuomotor control—have billions of parameters [44, 161, 32, 28, 312]. These models are not only slow to run, but also require heavy-duty hardware that is difficult to attach to edge devices such as mobile robots, adding even more overhead for remote inference. Edge hardware will improve over time, but as robot datasets grow in size, so will the best VLAs [152].

Thus, applying large models to real-time control problems effectively will require some form of asynchronicity: that is, a model must think about its future actions while executing a previous one. Action chunking [375, 168, 62], where a model outputs and executes a sequence of multiple actions for each inference call, presents a partial solution. Although action chunking has already achieved many state-of-the-art results in dexterous manipulation [32, 28, 312], it still suffers from the latency problem. Chunking sacrifices the reactivity of a system to external stimuli and also introduces discontinuities in the transition points between chunks, as adjacent chunks may jump between different modes (or "strategies") from the learned action distribution. Such anomalies are especially harmful to learning-based systems, as they produce a distribution shift in dynamics that the model is likely not equipped to handle. Naive smoothing strategies, such as averaging multiple predictions together [375], are not guaranteed to produce valid actions and may only make matters worse (e.g., see Figure 4.2).

A good real-time system must produce a consistent and continuous control signal, incor-

porating the latest observations without perturbing the environment's natural dynamics or the model's ability to produce correct actions. In this chapter, we present **real-time chunking (RTC)**, which poses asynchronous action chunking as an inpainting problem. Our algorithm generates the next action chunk while executing the previous one, freezing the actions that are guaranteed to be executed (due to inference delay) and "inpainting" the rest. It is applicable to any diffusion- [120] or flow-based [192] VLA, and operates purely at inference time, requiring no changes to existing training recipes.

Our contributions are as follows. First, we present a novel system for asynchronous, real-time inference of action chunking diffusion- or flow-based policies for continuous control. Since standard simulation benchmarks are quasi-static—and have mostly been saturated with pseudo open-loop inference strategies [62]—we devise a new benchmark based on the Kinetix simulator [221] consisting of 12 highly dynamic manipulation and locomotion tasks. In the real world, we evaluate RTC on 6 challenging bimanual manipulation tasks using the $\pi_{0.5}$ VLA [134] as the base policy. Across both simulation and the real world, we demonstrate that RTC is fast and performant; it is uniquely robust to inference latency, even in highly precise tasks such as lighting a match (Figure 4.1), and it achieves greatly improved task throughput on all real tasks.

4.2 Related Work

Action chunking, VLAs, and cascade control. Inspired in part by human motor control [168], action chunking has recently emerged as the de facto standard in imitation learning for visuomotor control [375, 62]. Learning to generate action chunks from human data requires expressive statistical models, such as variational inference [375, 101], diffusion [62, 63, 374, 375, 249, 315], flow matching [32, 38], vector quantization [172, 18, 228], or byte-pair encoding [253]. Recently, some of these methods have been scaled to billions of parameters, giving rise to VLAs [43, 65, 161, 32, 378, 60, 51, 376, 134, 253, 199], a class of large models built on pre-trained vision-language model backbones. With the capacity to fit ever-growing robot datasets [65, 157, 328, 91, 219, 144], as well as Internet knowledge from vision-language pre-training, VLAs have achieved impressive results in generalizable robot manipulation. When applied to real-world robots, action chunking policies are often used in conjunction with a lower-level, higher-frequency control loop—such as a PID controller—which translates the outputs of the policy (e.g., joint positions) to hardware-specific control signals (e.g., joint torques). In these cases, action chunking policies can be viewed as a form of cascade control [67], with the learned policy acting the outermost control loop. However, this is not always the case: for example, our simulated experiments use learned policies that output torques and forces directly. As such, we defer any exploration of the intersection between cascade control theory and learned action chunking policies to future work.

Reducing inference latency. A natural approach to improve the real-time capabilities

of a model is to simply speed it up. For instance, consistency policy [259] distills diffusion policies to elide expensive iterative denoising. Streaming diffusion policy [126] proposes an alternative training recipe that allows for very few denoising steps per controller timestep. Kim, Finn, and Liang [159] augment OpenVLA [161] with parallel decoding to elide expensive autoregressive decoding. More broadly, there is a rich literature on optimizing inference speed, both for diffusion models [276, 200, 301, 96] and large transformers in general [167, 135, 189]. Unfortunately, these directions cannot reduce inference cost below one forward pass. So long as this forward pass takes longer than the controller's sampling period, other methods will be needed for real-time execution.

Inpainting and guidance. There is a rich literature on image inpainting with pre-trained diffusion and flow models [257, 300, 203, 220]. In our work, we incorporate one such method [257] into our novel real-time execution framework with modifications (namely, soft masking and guidance weight clipping) that we find necessary for our setting. For sequential decision-making, Diffuser [140] pioneered diffusion-based inpainting for following state and action constraints in long-term planning, though their inpainting method is not guidance-based. (See Appendix C.4 for a comparison to the inpainting method from Diffuser applied to our setting.) Diffuser and other work [332, 8] have also guided diffusion models with value functions to solve reinforcement learning (RL) problems. Our work is distinct in that it is the first to apply either inpainting or guidance to real-time control.

Real-time execution. Real-time control has been studied long before the advent of VLAs. Similar to action chunking, model predictive control (MPC; [269]) generates plans over a receding time horizon; like our method, it parallelizes execution and computation, and uses the prior chunk to warm-start planning for the next. Though recent works combining learning methods with MPC have demonstrated real-time control capabilities in narrow domains [277, 113], they rely on explicit, hand-crafted dynamics models and cost functions. These methods are not applicable to our setting, which considers model-free imitation learning policies and tests them on unstructured, open-world manipulation tasks. Separately, in reinforcement learning, a variety of prior works have developed time-delayed decision-making methods [308, 95, 280, 330, 343, 346]. However, these approaches are not always applicable to imitation learning, and none of them leverage action chunking. Most recently, hierarchical VLA designs [312, 28] have emerged where the model is split into a System 2 (high-level planning) and System 1 (low-level action generation) component. The System 2 component contains the bulk of the VLA's capacity and runs at a low frequency, while the System 1 component is lightweight and fast. This approach is orthogonal to ours, and comes with its own tradeoffs (e.g., limiting the size of the System 1 component and requiring its own training recipe).

Bidirectional Decoding. The most closely related prior work is Bidirectional Decoding (BID; [201]), which enables fully closed-loop control with pre-trained action chunking policies via rejection sampling. While Liu et al. [201] do not consider inference delay, the BID algorithm can be used to accomplish the same effect as our guidance-based inpainting. We

compare to BID in our simulated benchmark, finding that it underperforms RTC while using significantly more compute.

4.3 Preliminaries and Motivation

We begin with an action chunking policy denoted by $\pi(\mathbf{A}_t|\mathbf{o}_t)$, where $\mathbf{A}_t = [\mathbf{a}_t, \mathbf{a}_{t+1}, ..., \mathbf{a}_{t+H-1}]$ is a chunk of future actions, $\mathbf{o}_t$ is an observation, and t indicates a controller timestep. We call H the *prediction horizon*. When action chunking policies are rolled out, only the first $s \leq H$ actions from each chunk are executed. We call s the *execution horizon*; often it is shorter than the prediction horizon, but still much greater than 1 (e.g., $s \approx H/2$ [62, 32, 134]). Chunked execution ensures temporal consistency at the expense of reactivity. A long execution horizon reduces a policy's responsiveness to new information, while a short one increases the likelihood of mode-jumping, jerky behavior resulting from discontinuities between chunks.

In this chapter, we consider policies trained with conditional flow matching [192], though our method can also be used with diffusion policies by converting them to flow policies at inference time [257, 98]. To generate an action chunk from a flow policy, random noise $\mathbf{A}_t^0$ is first sampled from a standard Gaussian, and then the flow's velocity field, $\mathbf{v}_\pi$ (a learned neural network) is integrated from $\tau = 0$ to 1 using the update rule

$$\mathbf{A}_t^{\tau+\frac{1}{n}} = \mathbf{A}_t^\tau + \frac{1}{n}\mathbf{v}_\pi(\mathbf{A}_t^\tau, \mathbf{o}_t, \tau), \tag{4.1}$$

where $\tau \in [0, 1)$ denotes a flow matching timestep, and n determines the number of denoising steps.

Now, let Δt be sampling period of the controller, i.e., the duration of a controller timestep, and let δ be the time it takes for the policy to generate an action chunk. We say that a system is *real-time* if it is guaranteed to produce a response (in our case: $\mathbf{a}_t$) to an event (receiving $\mathbf{o}_t$) within a fixed time constraint (Δt). If $\delta \leq \Delta t$, then meeting the real-time constraint is trivial, since an entire chunk can be generated between two controller timesteps. However, this is near impossible to achieve with modern VLAs. For example, with an RTX 4090 GPU, the 3 billion parameter π_0 VLA spends 46ms on the KV cache prefill alone, before any denoising steps [32], and targets a 50Hz control frequency ($\Delta t = 20$ms). Run in remote inference for mobile manipulation, π_0 lists 13ms of network latency, in perfect conditions with a wired connection. In a more realistic setting, the network overhead alone could easily exceed 20ms. Kim, Finn, and Liang [159], who optimize the 7B OpenVLA model [161] specifically for inference speed, achieve no better than 321ms of latency on a server-grade A100 GPU.

Naive synchronous inference, the default in many prior works [32, 161, 44, 134, 159, 315], simply starts inference at the end of the execution horizon and waits while the policy generates

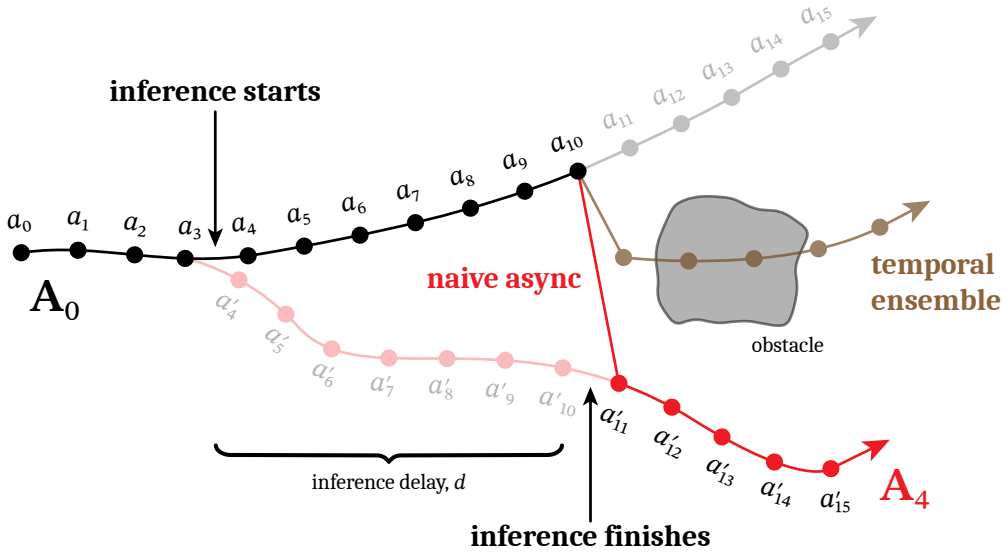


Figure 4.2: An illustration of a typical bifurcation between consecutive chunks. Inference is started between timesteps 3 and 4. The original chunk that was executing, $\{a_t\}$ (black), had planned to go above the obstacle while the newly generated chunk $\{a'_t\}$ (red) goes below the obstacle. However, $\{a'_t\}$ is not available until $d = 7$ steps later. A naive asynchronous algorithm might jump from a_{10} to a'_{11}, inducing a very high, out-of-distribution acceleration. Temporal ensembling [375], i.e., interpolating between chunks, reduces the acceleration but produces poor actions.

the next chunk. When $\delta > \Delta t$, this introduces visible pauses between chunks that not only slow down execution but also change the dynamics of the robot, introducing distribution shift between training and evaluation. To develop a real-time strategy, we must first introduce *asynchronous* inference, where inference is started early and happens concurrently with execution.

We define $d := \lfloor \delta / \Delta t \rfloor$ and call this quantity the *inference delay*, corresponding to number of controller timesteps between when $\mathbf{o}_t$ is received and when $\mathbf{A}_t$ is available.[1] Let $\mathbf{a}_{t'|t}$ denote the $(t' - t)$-th action of chunk $\mathbf{A}_t$, generated from observing $\mathbf{o}_t$. If $\mathbf{A}_0$ is currently executing, and we desire an execution horizon of s, then an asynchronous algorithm must start inference at $s - d$. So long as $d \leq H - s$, then this strategy will satisfy the real-time constraint and guarantee that an action is always available when it is needed. However, since the policy cannot know what will happen between steps $s - d$ and s while generating $\mathbf{A}_{s-d}$, the transition point between $\mathbf{a}_{s-1|0}$ and $\mathbf{a}_{s|s-d}$ may be arbitrarily discontinuous and out-of-distribution. Similar to a too-short execution horizon, this strategy leads to jerky

[1] For simplicity, we do not consider delays or synchronization issues at the sub-timestep level; we assume that the environment or lower-level controller provides $\mathbf{o}_t$ at the same instant that $\mathbf{a}_{t-1}$ is consumed.

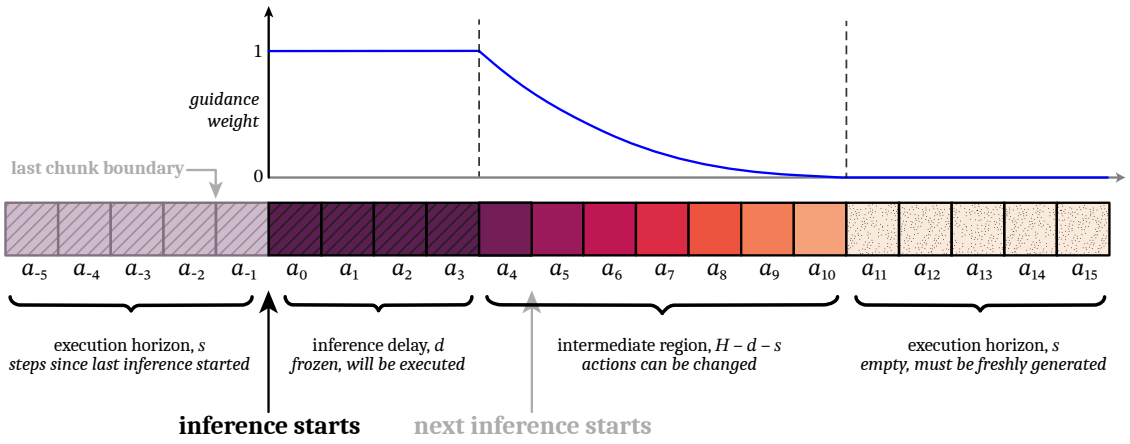


Figure 4.3: A diagram illustrating how action generation attends to the previous action chunk in real-time chunking. If inference starts after the execution of a_{-1} and the inference delay is $d = 4$, then the newly generated chunk will not be available until after a_3 is consumed. Therefore, $a_{0:3}$ are "frozen" and are attended to with a full guidance weight of 1. In the intermediate region, $a_{4:10}$, actions from the previous chunk are available but may be updated, since inference will have finished before a_4 is needed. This region is attended to with an exponentially decreasing guidance weight. Finally, the last $s = 5$ actions are beyond the end of the previous chunk, and need to be freshly generated. The execution horizon, s, is a hyperparameter constrained by $d \leq s \leq H - d$.

behavior that is worsened dramatically with higher delays; see Figure 4.2.

4.4 Real-Time Chunking via Inpainting

The key challenge in real-time execution is to maintain continuity between chunks. By the time a new chunk is available, the previous one has already been executed partway, and therefore the new chunk must be "compatible" with the previous one. At the same time, the new chunk should still incorporate new observations, so that the policy does not lose the ability to react and make corrections.

Our key insight is to pose real-time chunking as an inpainting problem. To make the new chunk "compatible", we must use the overlapping timesteps where we have access to the remaining actions of the previous chunk. The first d actions from the new chunk cannot be used, since those timesteps will have already passed by the time the new chunk becomes available. Thus, it makes sense to "freeze" those actions to the values that we know *will* be executed; our goal is then to fill in the remainder of the new chunk in a way that is consistent with this frozen prefix (see Figure 4.3), much like inpainting a section of an image that has been removed. We describe this basic inpainting principle in Sec. 4.4. In Sec. 4.4,

we introduce a *soft masking* extension that is critical for full cross-chunk continuity; finally, we describe our full real-time chunking system in Sec. 4.4.

Inference-Time Inpainting with Flow Matching

Inpainting is a known strength of iterative denoising frameworks such as diffusion and flow matching. We build on the training-free image inpainting algorithm from Pokle et al. [257], which is itself based on pseudoinverse guidance (ΠGDM; [300]). The algorithm operates by adding a gradient-based guidance term to the learned velocity field $\mathbf{v}$ at each denoising step (Equation 4.1) that encourages the final generation to match some target value, $\mathbf{Y}$, which is a corrupted version of the desired result. In the case of image inpainting, the corruption operator is masking, $\mathbf{Y}$ is the masked image, and the desired result is a full image consistent with $\mathbf{Y}$ in the non-masked areas. The ΠGDM gradient correction, specialized to our setting, is given by

$$\mathbf{v}_{\Pi\text{GDM}}(\mathbf{A}_t^\tau, \mathbf{o}_t, \tau) = \mathbf{v}(\mathbf{A}_t^\tau, \mathbf{o}_t, \tau) + \min\left(\beta, \frac{1-\tau}{\tau \cdot r_\tau^2}\right)\left(\mathbf{Y} - \widehat{\mathbf{A}_t^1}\right)^\top \text{diag}(\mathbf{W})\, \frac{\partial \widehat{\mathbf{A}_t^1}}{\partial \mathbf{A}_t^\tau} \tag{4.2}$$

$$\text{where } \widehat{\mathbf{A}_t^1} = \mathbf{A}_t^\tau + (1-\tau)\mathbf{v}(\mathbf{A}_t^\tau, \mathbf{o}_t, \tau), \tag{4.3}$$

$$r_\tau^2 = \frac{(1-\tau)^2}{\tau^2 + (1-\tau)^2}. \tag{4.4}$$

$\widehat{\mathbf{A}_t^1}$ is an estimate of the final, fully denoised action chunk and $\mathbf{W}$ is the mask. We are abusing notation by treating $\mathbf{Y}$, $\mathbf{A}_t$, and $\mathbf{W}$ as vectors of dimension HM where M is the dimension of each action. Thus, the guidance term is a vector-Jacobian product and can be computed using backpropagation. The guidance weight clipping, β, is our addition; we found that without it, the algorithm became unstable with the small number of denoising steps commonly used in control problems (see C.2 for an ablation).

Soft Masking for Improved Cross-Chunk Continuity

In practice, naively inpainting using only the first d timesteps of the previous action chunk is often insufficient to ensure that the new chunk takes a consistent strategy, particularly when d is small (e.g., see Figure 4.4). The ΠGDM correction is not perfect, and a small d leads to a weak guidance signal, which can allow for the new chunk to still switch strategies and cause discontinuities. Our solution, illustrated in Figure 4.3, is to give our policy more cross-chunk continuity by considering not just the first d overlapping actions, but all $H - s$ overlapping actions. We do this via *soft masking*, setting $\mathbf{W}$ to real-valued weights rather than 1s and 0s. The first d actions get a weight of 1; the last s actions of the new chunk do not overlap with the previous chunk, so they get a weight of 0; the actions in between get weights that exponentially decay from 1 to 0, accounting for the fact that actions further in the future

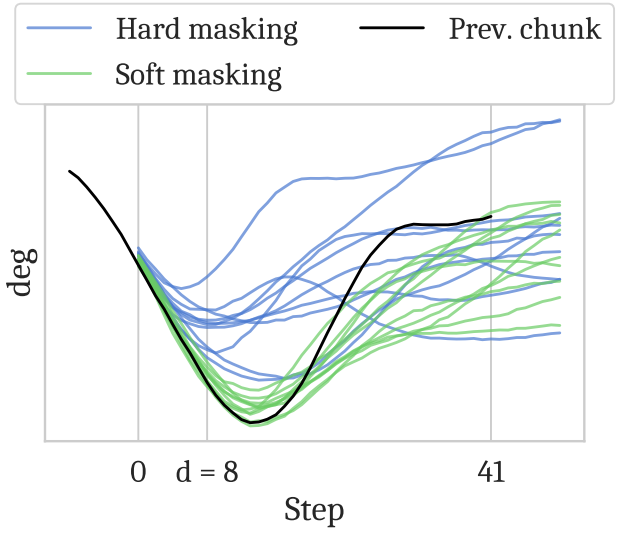


Figure 4.4: A comparison of naive inpainting (hard masking) and our proposed soft masking method: note that hard masking does not match the frozen region very well and produces faster changes in direction.

should be treated with more uncertainty. The resulting expression for $\mathbf{W}$ is given by

$$\mathbf{W}_i = \begin{cases} 1 & \text{if } i < d \\ c_i \frac{e^{c_i}-1}{e-1} & \text{if } d \le i < H-s \\ 0 & \text{if } i \ge H-s \end{cases} \quad \text{where } c_i = \frac{H-s-i}{H-s-d+1}, \quad i \in \{0, \dots, H-1\}. \tag{4.5}$$

Intuitively, $\mathbf{W}$ modulates the "attention" paid to each corresponding action from the previous chunk. See Appendix C.4 for a comparison between different decay schedules.

Real-Time Chunking

We present our full real-time chunking system in Algorithm 1 (complemented by Figure 4.3). The controller interfaces with our algorithm via GetAction, which is called every Δt to consume an action $\mathbf{a}_{t-1}$ and provide the next observation $\mathbf{o}_t$. The InferenceLoop runs in a background thread so that an action is always available. It forecasts the next delay, d, by keeping a buffer of past delays. The execution horizon, s, can change from chunk to chunk; the user provides a minimum desired horizon, $s_{\min}$, and the actual horizon for a given chunk is $\max(d, s_{\min})$ where d is the delay encountered when computing the *next* chunk. Finally, the algorithm describes the inpainting with soft masking procedure in GuidedInference, which explicitly defines a denoising function (Eq. 4.3) and computes a vector-Jacobian product, which can be done with reverse-mode autodifferentiation [16].

Algorithm 1 Real-Time Chunking

Require: flow policy π with prediction horizon H, minimum execution horizon $s_{\min}$, mutex $\mathcal{M}$, condition variable $\mathcal{C}$ associated with $\mathcal{M}$, initial chunk $\mathbf{A}_{\text{init}}$, initial delay estimate d_{init}, delay buffer size b, number of denoising steps n, maximum guidance weight β

procedure INITIALIZESHAREDSTATE ▷ Initialize mutex-protected shared variables
    $t = 0$; $\mathbf{A}_{\text{cur}} = \mathbf{A}_{\text{init}}$, $\mathbf{o}_{\text{cur}} = \text{null}$

function GETACTION($\mathbf{o}_{\text{next}}$) ▷ Called at an interval of Δt by controller
    with $\mathcal{M}$ acquired **do**
        $t = t + 1$
        $\mathbf{o}_{\text{cur}} = \mathbf{o}_{\text{next}}$
        notify $\mathcal{C}$
        return $\mathbf{A}_{\text{cur}}[t-1]$

4.5 Experiments

In our experiments, we aim to answer the following questions. First, how does RTC compare to existing methods in highly dynamic and stochastic environments, and under increasing inference delays? Second, how important is soft masking (Sec. 4.4) to RTC? Third, how does RTC affect the performance *and* speed of real-world dexterous robots?

We first evaluate RTC using a benchmark of 12 highly dynamic and stochastic environments in the Kinetix [221] simulator. We use this benchmark to compare the performance of RTC to other methods under simulated inference delays, as well as investigate the effect of soft masking. Then, using the $\pi_{0.5}$ VLA [134] as the base model, we evaluate the performance and speed of RTC on 6 challenging bimanual dexterous manipulation tasks, including 2 mobile manipulation tasks.

Simulated Benchmark

Most simulated imitation learning benchmarks are quasi-static, and standard chunked execution with a long enough execution horizon can achieve near-perfect success rates [62]. We instead create a benchmark of 12 dynamic tasks in Kinetix [221], which uses force-based control, so inference delay *necessitates* asynchronous execution (there is no concept of "holding position"). We select 10 existing environments and create 2 new ones such that all environments involve dynamic motions like throwing, catching, and balancing. To simulate imperfect actuation, we add Gaussian noise to the actions, making closed-loop corrections crucial for success.

Setup. To generate data for imitation learning, we first train expert policies using RPO

procedure INFERENCELOOP ▷ Run inference in a looping background thread
  acquire $\mathcal{M}$
  $\mathcal{Q}$ = new Queue([d_{init}], maxlen=b) ▷ Holds a limited buffer of past inference delays
  loop
    wait on $\mathcal{C}$ until $t \geq s_{\min}$
    $s = t$ ▷ s is the number of actions executed since last inference started
    $\mathbf{A}_{\text{prev}} = \mathbf{A}_{\text{cur}}[s, s+1, \ldots, H-1]$ ▷ Remove the s actions that have already been executed
    $\mathbf{o} = \mathbf{o}_{\text{cur}}$
    $d = \max(\mathcal{Q})$ ▷ Estimate the next inference delay conservatively
    with $\mathcal{M}$ released **do**
      $\mathbf{A}_{\text{new}} = \text{GUIDEDINFERENCE}(\pi, \mathbf{o}, \mathbf{A}_{\text{prev}}, d, s)$

    $\mathbf{A}_{\text{cur}} = \mathbf{A}_{\text{new}}$ ▷ Swap to the new chunk as soon as it is available
    $t = t - s$ ▷ Reset t so that it indexes into $\mathbf{A}_{\text{new}}$
    enqueue t onto $\mathcal{Q}$ ▷ Record the observed delay

function GUIDEDINFERENCE($\pi, \mathbf{o}, \mathbf{A}_{\text{prev}}, d, s$)
  compute $\mathbf{W}$ using Eq. 4.5; right-pad $\mathbf{A}_{\text{prev}}$ to length H; initialize $\mathbf{A}^0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  for $\tau = 0$ to 1 with step size $1/n$ **do**
    $f_{\widehat{\mathbf{A}^1}} = \mathbf{A}' \mapsto \mathbf{A}' + (1-\tau)\mathbf{v}_\pi(\mathbf{A}', \mathbf{o}, \tau)$ ▷ Define denoising function (Eq. 4.3)
    $\mathbf{e} = \left(\mathbf{A}_{\text{prev}} - f_{\widehat{\mathbf{A}^1}}(\mathbf{A}^\tau)\right)^\top \text{diag}(\mathbf{W})$ ▷ Weighted error term from Eq. 4.2
    $\mathbf{g} = \mathbf{e} \cdot \frac{\partial f_{\widehat{\mathbf{A}^1}}}{\partial \mathbf{A}'}\Big|_{\mathbf{A}'=\mathbf{A}^\tau}$ ▷ Compute vector-Jacobian product from Eq. 4.2 via autodiff
    $\mathbf{A}^{\tau+\frac{1}{n}} = \mathbf{A}^\tau + \frac{1}{n}\left(\mathbf{v}_\pi(\mathbf{A}^\tau, \mathbf{o}, \tau) + \min\left(\beta, \frac{1-\tau}{\tau \cdot r_\tau^2}\right)\mathbf{g}\right)$ ▷ Integration step (Eq. 4.1)
  return $\mathbf{A}^1$

[268] and a binary success reward. For each environment, we train 6 expert policies with different seeds and then generate a 1M transition dataset with a different policy selected each episode. We then train action chunking flow policies with a prediction horizon of $H = 8$ and a 4-layer MLP-Mixer [321] architecture for 32 epochs. We report binary success rates with 2048 rollouts per data point, and simulate delays between 0 (fully closed-loop) and 4 (the maximum supported when $H = 8$).

Baselines. We compare against the following baselines:

- *Naive async.* This strategy does not pay attention to the previous action chunk at all when generating a new one, naively switching chunks as soon as the new one is ready.
- *Bidirectional decoding (BID; [201]).* This strategy uses rejection sampling to keep continuity across chunks. We use a batch size of $N = 32$, mode size of $K = 3$, and a checkpoint trained for 8 epochs as the weak policy.

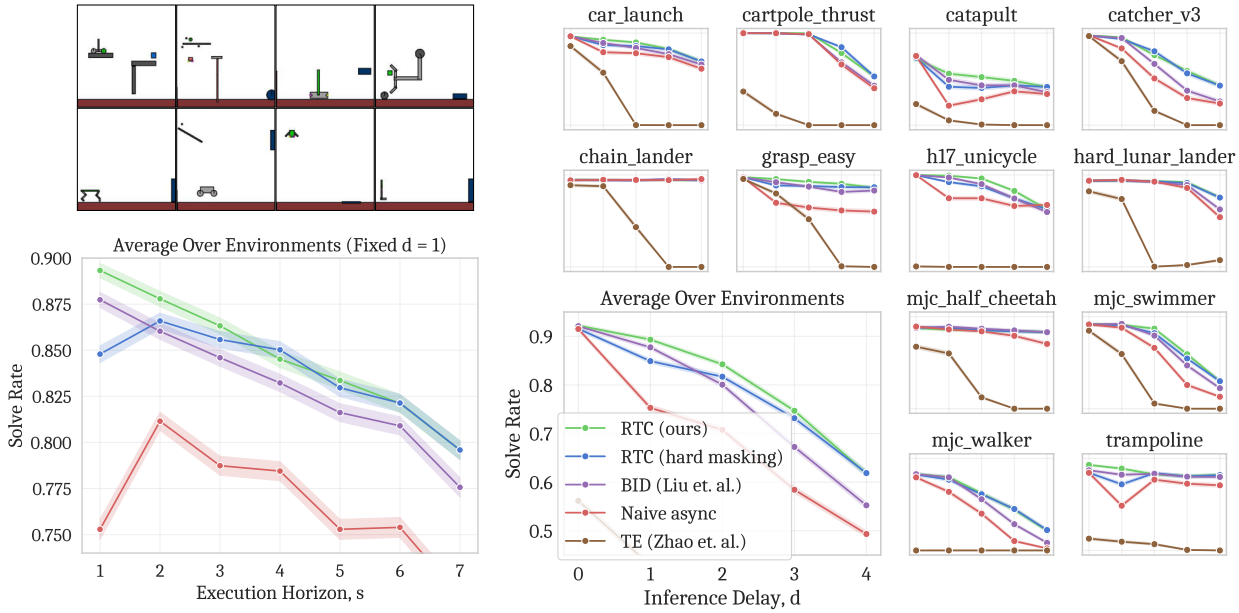


Figure 4.5: **Top left:** Kinetix environments; each involves getting a green object on the left to touch a blue one on the right. **Bottom left:** Execution horizon vs. solve rate with a fixed inference delay of 1. Only RTC and BID take full advantage of faster updates, showing strictly increasing performance with decreasing execution horizon. **Right:** Inference delay vs. solve rate with a fixed execution horizon of $s = \max(d, 1)$. RTC outperforms all baselines. Furthermore, soft masking (Sec. 4.4) improves performance at lower inference delays and execution horizons. Each data point represents 2048 trials, and 95% Wilson score intervals are shaded in.

- *Temporal ensembling (TE; [375]).* This strategy involves keeping a buffer of predicted action chunks and executing an average of all actions predicted for a particular timestep.

Results. Figure 4.5 shows the simulated results. In the delay plots (right): TE performs poorly across the board, even with an inference delay of $d = 0$, illustrating the multi-modality of our benchmark—averages of valid actions are not necessarily valid. RTC shows the most robustness to inference delays, outperforming BID, and the gap widens with increasing delay; note that BID uses significantly more compute than RTC by sampling batches of 64 action chunks, 32 from a strong model and 32 from a weak model. Additionally, we find that hard masking somewhat underperforms soft masking, particularly when d is smaller, supporting our claims in Sec. 4.4. Finally, in the execution horizon plot (left), we find that thanks to its continuity across chunks, RTC is better able to take advantage of closed-loop corrections, always performing better with a decreasing execution horizon.

Real-World Results

Next, we deploy our full real-time chunking system to the real world. We use the $\pi_{0.5}$VLA [134] as our base policy, and evaluate RTC on a bimanual system with two 6-DoF arms and parallel jaw grippers. Unlike our simulated benchmark, the robots use position control, and so synchronous inference—stopping between chunks—is a reasonable default strategy, used in many prior works [32, 134, 159, 253]. Our goal is to improve upon synchronous inference in a combination of both performance *and* speed.

Setup. We use $\pi_{0.5}$ ($H = 50$, $\Delta t = 20$ms) with $n = 5$ denoising steps, giving a model latency of 76ms for the baselines and 97ms for RTC. We use remote inference over LAN, which adds 10-20ms of latency, giving a starting inference delay around $d \approx 6$ for RTC. However, we would like to understand how the system behaves with higher inference latencies, simulating, e.g., scaling up the model size or running inference on a distant cloud server. Thus, we also evaluate all methods with +100ms and +200ms of injected latency, corresponding to $d \approx 11$ and $d \approx 16$, respectively.

Tasks and scoring. Each episode gets an integer score corresponding to how many substeps of the task it completed successfully. We evaluate the following tasks:

- *Light candle (5 steps, 40s cutoff).* Pick up a match and matchbox, strike the match, use it to light a candle, and drop it in a bowl.
- *Plug ethernet (6 steps, 120s cutoff).* Pick up the end of an ethernet cable, reorient it, plug it into a server rack, and repeat the process for the other end.
- *Make bed, mobile (3 steps, 200s cutoff).* Move the corner of a blanket and 2 pillows from the foot to the head of a bed.
- *Shirt folding (1 step, 300s cutoff).* Fold a shirt from a flattened position.
- *Batch folding (4 steps, 300s cutoff).* Take a varied, crumpled clothing item out of a bin, flatten it, fold it, then place it neatly on a pile.
- *Dishes in sink, mobile (8 steps, 300s cutoff).* Move 4 varied items from a counter into a sink.

See the accompanying blog post for videos of each task. We evaluate each task and method for 10 trials for a total of 480 episodes, adding up to 28 hours of pure robot execution time. We also post-hoc annotate the score for each episode and the timestamp at which each step is achieved.

Baselines. We compare to the following baselines:

- *Synchronous.* This corresponds to the default inference strategy in prior work [32, 134, 159, 253], which executes $s = 25$ actions and then pauses while the new chunk is generated.
- *TE, sparse.* This is similar to *naive async* in our simulated results; it executes $s = 25$ actions at a time while computing the next chunk in parallel. We found it significantly

reduced jerkiness to also apply TE, even though only the first $H - s - 2d$ executed steps of each chunk have overlapping actions to ensemble.

- *TE, dense.* This strategy is the closest to the original TE in Zhao et al. [375]. We run inference as often as possible, resulting in $s = d$ for every chunk. This results in there always being at least 2 overlapping action chunks to ensemble, and often more.

We do not compare to BID [201] in the real world, as we found in simulation that it underperforms RTC while using significantly more compute—when applied to $\pi_{0.5}$with a batch size of 16, BID has 2.3 times the latency of our method (see C.3 for latency measurements).

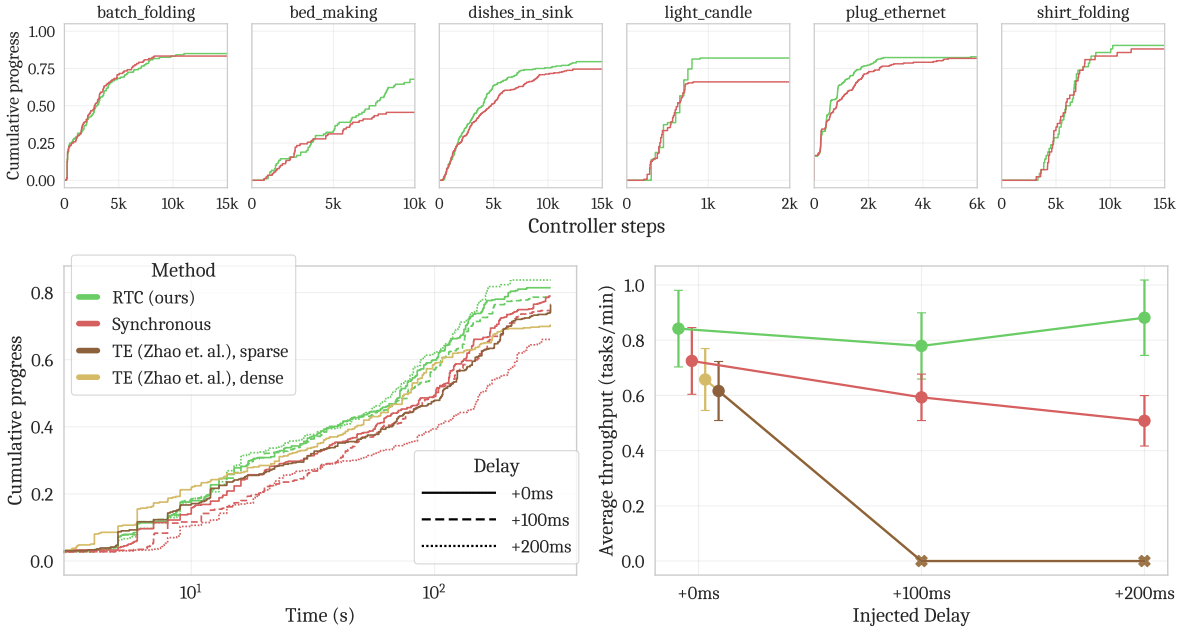


Figure 4.6: **Top:** Controller steps (equivalent to elapsed time with inference pauses removed multiplied by 50Hz) vs. cumulative progress for each task, aggregated across all delays. Progress is measured in discrete steps corresponding to the subsections of each task. **Left:** Time (including inference pauses) vs. cumulative progress aggregated across all tasks. The x-axis is log scale to better show progress during both short and long-horizon tasks. **Right:** Inference delay vs. average throughput, defined as the proportion of task completed divided by duration of episode averaged over episodes. Error bars are ±1 SEM. Average throughput gives a balanced view of both speed and performance for each method. Neither TE variant can run at +100 or +200ms of injected latency, causing such high oscillations that the robot's protective stop is triggered.

Results. We present the results in Figure 4.6. In average task throughput, a measurement of both speed and performance, RTC achieves the best score at all inference delays with a statistically significant result at +100 and +200ms. RTC is completely robust to injected

delay, showing no degradation, whereas synchronous degrades linearly and both TE variants do not run at all due to causing such high oscillations that the robot's protective stop is triggered (see videos). Inspecting the per-task results (Figure 4.5, top), we can conclude that RTC helps with more than just execution speed: it completes tasks faster than synchronous inference **even when inference pauses are removed**. All tasks, except for *light candle*, allow for retrying until the time limit (and $\pi_{0.5}$ does, in general, exhibit robust retrying behavior). Even though synchronous inference often reaches a similar final score, RTC often completes more of the task earlier in the episode, reflecting fewer mistakes and less retrying. In *light candle*, the most precision-sensitive task—and also the only one without retrying—RTC shows a large advantage in final score, reflecting a higher overall success rate. Interestingly, the same is true in *bed making*, even though that task does elicit retrying. The policy particularly struggles to manipulate the pillows, and *bed making* is the hardest task overall, which may be why RTC has a strong effect.

4.6 Discussion

Real-time chunking is an inference-time algorithm for asynchronous execution of action chunking policies that demonstrates speed and performance across simulation and real-world experiments, including under significant inference delays. However, this work is not without limitations: it adds significant computational overhead compared to methods that sample directly from the base policy, and it is applicable only to diffusion- and flow-based policies. Additionally, while our real-world experiments cover a variety of challenging manipulation tasks, there are more dynamic settings that could benefit even more from real-time execution. One example is legged locomotion, which is represented in our simulated benchmark but not our real-world results.

Chapter 5

Training-Time Action Conditioning for Efficient Real-Time Chunking

5.1 Introduction

Unlike chatbots or search engines, embodied agents must operate in real time. The feedback loop between an agent's actions and its environment necessitates reactivity — like a human athlete, an agent cannot simply "stop and think" while the outside world changes. However, the ever-increasing size of frontier models makes this more and more difficult. Nowhere is this more evident than in the domain of robot learning, where vision-language-action models (VLAs) consisting of billions of parameters have increasingly been used to control robots at high frequencies to accomplish dexterous tasks. Producing smooth yet reactive trajectories when the model inference latency is in the tens to hundreds of milliseconds is no small challenge.

Real-time chunking (RTC; [29]) presents an approach to this problem that combines action chunking [62, 374], flow matching [192], and inference-time inpainting [300, 257]. In RTC, action chunks are predicted asynchronously — the next chunk is generated while the current one is still executing. To ensure continuity between chunks, each generation is conditioned on a frozen *prefix* of previously predicted actions, inpainting the rest. However, the inference-time inpainting method used by RTC introduces additional computational overhead — and hence latency — that somewhat defeats the purpose of a real-time execution framework. Empirically, we also find that inference-time inpainting is fundamentally limited in its ability to handle high inference delays.

In this work, we augment RTC with an inpainting method that simulates inference delay at training time and eliminates any inference-time computational overhead. Our method works as a drop-in replacement for inference-time RTC: it requires no modifications to the model

architecture or the robot runtime, and can be implemented with only a few additional lines of code. On simulated benchmarks, we find that training-time RTC outperforms inference-time RTC at higher inference delays. In the real world, we demonstrate that training-time RTC can be successfully added by fine-tuning a base model that was *not* pre-trained with action prefix conditioning. By applying training-time RTC to the $\pi_{0.6}$ VLA [318], we show improved performance over inference-time RTC on two highly complex tasks: box building and espresso making.

5.2 Related Work

Action chunking and VLAs. Action chunking [375, 62] is the de facto standard in end-to-end imitation learning for visuomotor control. Recently, augmenting vision-language models (VLMs) to produce action chunks has demonstrated great success in robot manipulation, giving rise to vision-language-action models (VLAs) [43, 65, 161, 32, 378, 60, 51, 376, 134, 253, 199]. Subsequently, a plethora of methods have emerged to address the tension between large VLAs and high-frequency control. For example, Gemini Robotics [312] and GR00T [28] employ hierarchical VLA designs where the model is split into a heavyweight System 2 (high-level planning) and lightweight System 1 (low-level action generation) component. MiniVLA [18] and SmolVLA [296] present VLA architectures that are altogether faster and more efficient than most designs, making inference at the edge more feasible. These contributions are orthogonal to ours, and come with their own tradeoffs (e.g., modified network architectures and training recipes).

Real-time execution of VLAs. The most closely related prior work is real-time chunking (RTC; [29]), which introduces an asynchronous execution framework that serves as a foundation for this work. Also related is SmolVLA [296], which presents an asynchronous execution algorithm that is similar to that of RTC; however, SmolVLA does not solve the inter-chunk discontinuity problem, which leads to out-of-distribution "jerks" between chunks. Concurrently to this work, A2C2 [281] and VLASH [309] both solve the discontinuity problem by adding a lightweight correction head and by conditioning on a single future action, respectively. In contrast to VLASH, we condition on a full prefix of future actions.

5.3 Preliminaries

We use the same problem formulation as RTC [29]: we begin with an action chunking policy denoted by $p(\mathbf{A}_t|\mathbf{o}_t)$, where $\mathbf{A}_t = [\mathbf{a}_t, \mathbf{a}_{t+1}, ..., \mathbf{a}_{t+H-1}]$ is a chunk of future actions, $\mathbf{o}_t$ is an observation, and t indicates a controller timestep. We call H the *prediction horizon*, and at inference time, we roll out each chunk for $s \leq H$ timesteps, where s is the *execution horizon*.

To account for model inference, we define the quantity d to be the inference delay in units of

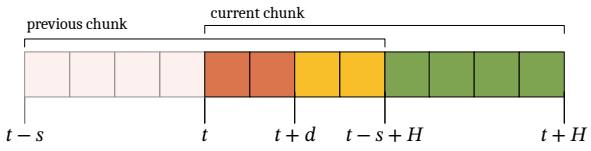


Figure 5.1: A diagram illustrating two overlapping action chunks. The d actions between t and $t + d$, taken from the previous chunk, are the action prefix (red). From the diagram, we can easily see that we must satisfy the constraint $t + d \leq t - s + H \rightarrow d \leq H - s$ to have a valid action prefix. Note that inference-time RTC uses all $H - s$ overlapping actions (red and yellow) to guide the generation of the current chunk, whereas training-time RTC only uses the first d actions (red).

controller timesteps. If inference begins at step t, then the resulting action chunk will not be available until step $t + d$, and so the first d actions cannot actually be executed. However, so long as $d \leq H - s$, these first d timesteps will correspond to actions from the previous chunk that can be executed in the meantime. We call these d actions from the previous chunk that overlap with the current chunk the *action prefix* (see Figure 5.1).

We consider policies trained with conditional flow matching [192], which minimizes the following loss:

$$\mathbf{A}_t^\tau = \tau \mathbf{A}_t + (1 - \tau)\boldsymbol{\epsilon} \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{5.1}$$

$$\mathcal{L}(\theta) = \mathbb{E}\, ||\mathbf{v}_\theta(\mathbf{A}_t^\tau, \mathbf{o}_t, \tau) - (\boldsymbol{\epsilon} - \mathbf{A}_t)||^2 \tag{5.2}$$

where $\mathbf{v}_\theta$ is a neural network and τ denotes the flow matching timestep. At inference time, $\mathbf{v}_\theta$ can be integrated from $\tau = 0$ to 1 to produce samples from the dataset distribution $p(\mathbf{A}_t|\mathbf{o}_t)$.

5.4 Training-Time Action Conditioning

Inference-time RTC [29] conditions the policy on the action prefix (Figure 5.1, red) using a inference-time inpainting method based on pseudoinverse guidance [257, 300]. For improved continuity between chunks, inference-time RTC additionally conditions on *all* overlapping actions, using exponentially decreasing weights for actions beyond the prefix (Figure 5.1, yellow). In RTC, this is referred to as "soft masking". While pseudoinverse guidance affords great flexibility — enabling soft masking — it also requires computing a vector-Jacobian product (using backpropagation) during each denoising step.

The core insight of this work is that we can condition the policy on action prefixes at training time by simulating inference delay. While this does not afford the same flexibility as inference-time inpainting, it eliminates the computational overhead. Formally, we can learn $p(\mathbf{A}_{t+d:H}|\mathbf{o}_t, \mathbf{A}_{t:t+d})$, where $\mathbf{A}_{t:t+d}$ is an action prefix (Figure 5.1, red) and $\mathbf{A}_{t+d:H}$ is an action

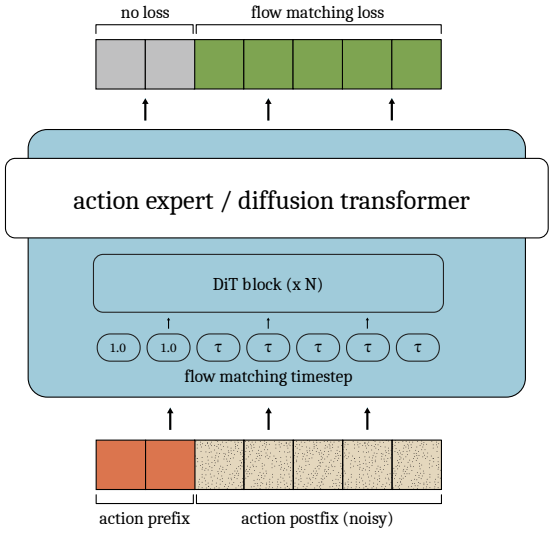


Figure 5.2: An illustration of our conditioning architecture, as applied to a standard diffusion transformer such as the $\pi_{0.6}$ action expert. We always feed in ground-truth, non-noisy prefix actions, while learning to denoise the postfix actions. The flow matching timestep differs between tokens, which indicates the inference delay to the model.

postfix (Figure 5.1, yellow and green), both taken from the same ground-truth action chunk. Implementing this for most standard policy architectures only requires 3 minimal changes:

1. Modify the model architecture to allow for a different flow matching timestep for each action timestep. For a diffusion-transformer-like architecture [250], which uses adaLN-zero conditioning for the flow matching timestep, this is trivial — simply allow the scale, shift, and gate to differ between tokens. This does not change the number of learnable parameters.

2. Use ground-truth, non-noisy actions for the prefix, and set the corresponding flow matching timesteps to 1. Do not change anything for the postfix. This conditions the model on the ground-truth action prefix while using it to denoise only the postfix.

3. Mask the loss function so that loss is only computed on outputs corresponding to the postfix.

See Figure 5.2 for an illustration of this conditioning scheme as applied to a standard diffusion-transformer-like achitecture (e.g., the $\pi_{0.6}$ action expert). See Algorithm 4 for Python code fully implementing loss calculation and action generation. In practice, since we do not know

the exact inference delay ahead of time (and inference delays in the real world may vary), we sample d randomly during training.

With these modifications, action generation takes as input an action prefix $\mathbf{A}_{t:t+d}$ and the delay itself d and produces as output an action postfix $\mathbf{A}_{t+d:H}$. As such, it adheres to the same interface as the action generation component of inference-time RTC (see [29], Algorithm 1) and thus acts as a seamless drop-in replacement.

5.5 Experiments

In our experiments, we aim to compare training-time RTC to inference-time RTC, as well as to naive synchronous and asynchronous baselines. Our simulated experiments use the same dynamic Kinetix [221] benchmark as RTC (see [29] for details). Our real-world experiments build on the $\pi_{0.6}$ base model [318], and include two precise and challenging tasks: box building and espresso making. We use the same experimental setup as $\pi^*_{0.6}$ [11].

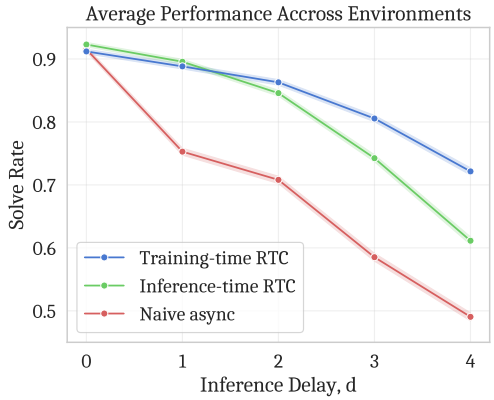


Figure 5.3: Simulated results: inference delay vs. solve rate with a fixed execution horizon of $s = \max(d, 1)$. Training-time RTC performs better than inference-time RTC at inference delays of 2 or higher. Each data point represents 2048 trials, and 95% Wilson score intervals are shaded in.

Simulated Results

In the dynamic Kinetix benchmark, following RTC [29], we train action chunking flow policies with a prediction horizon of $H = 8$ and a 4-layer MLP-Mixer [321] architecture for 32 epochs on data generated by a mixture of expert policies. We report binary success rates with 2048 rollouts per data point and test delays between 0 (fully closed-loop) and 4 (the maximum supported when $H = 8$). Naive asynchronous and inference-time RTC both use the same checkpoint, which is trained normally *without* action prefix conditioning for 32 epochs.

For training-time RTC, we resume training from the 24th epoch and fine-tune for 8 epochs with action prefix conditioning. We do this so that all methods are matched in training compute. We sample delays from $\{0, 1, 2, 3, 4\}$ with exponentially decreasing weights, as we found that higher delays need less training supervision. Better results could likely be obtained by spending more training compute training individual checkpoints for each delay.

The results are presented in Figure 5.3. We find that training-time RTC outperforms inference-time RTC at inference delays of 2 and higher — with the gap significantly widening as the delay increases. This is likely because, as the size of the prefix grows, the inpainting algorithm has to "work harder" to produce a consistent postfix. In these cases, the training-time algorithm is more robust than the pure inference-time algorithm, which relies on a linearization obtained from the Jacobian of the model. Training-time RTC performs very marginally worse at delays of 1 and 0, likely because training-time RTC does not always receive training supervision for every action — i.e., slightly less training compute is spent learning to generate the first and second actions.



(a) Box building task (b) Espresso making task

Figure 5.4: Real-world evaluation tasks: building a cardboard box and making espresso (including grinding, tamping, extracting, and pouring).

Real-World Results

In our real-world experiments, we use the $\pi_{0.6}$ base model [318] and test on the espresso making and box building tasks from $\pi^*_{0.6}$ [11]; see Figure 5.4 for an illustration. As in the simulated experiments, we use the same checkpoint for the synchronous baseline and inference-time RTC, and train a second checkpoint with action prefix conditioning for training-time RTC. Both checkpoints are fine-tuned from the base model on the target task for 8,000 gradient steps with a batch size of 512. We sample delays uniformly between 0 and 10 during training, which supports a maximum latency of 200ms on a 50Hz robot. During evaluations, we perform inference on a remote H100 server with 5 denoising steps, averaging 108ms of end-to-end latency for training-time RTC ($d \approx 5$) and 135ms for inference-time RTC ($d \approx 7$).

The results are presented in Figure 5.5. We find that training-time RTC maintains both performance and speed parity with inference-time RTC without any computational overhead. Both variants of RTC clearly improve speed over the synchronous inference baseline, which exhibits visible pauses in between chunks.

5.6 Discussion and Future Work

In this work, we have presented a simple and effective drop-in replacement for real-time chunking (RTC) that elides any inference-time computational overhead by adding a small amount of additional training compute. Our method requires no modifications to the model architecture or the robot runtime, and can be implemented with only a few additional lines of code. Our simulated experiments show that training-time RTC outperforms inference-time RTC at higher inference delays, while our real-world experiments show that training-time RTC maintains both performance and speed parity with inference-time RTC without any computational overhead.

However, training-time RTC is fundamentally less flexible than inference-time RTC; it only supports conditioning on a "hard" action prefix corresponding to the inference delay, whereas inference-time RTC can "softly" incorporate additional actions beyond the prefix. Additionally, training-time RTC requires carefully choosing the distribution of delays to simulate at training time based on the expected inference latency. We look forward to future work that can address these limitations and incorporate the best of both worlds.

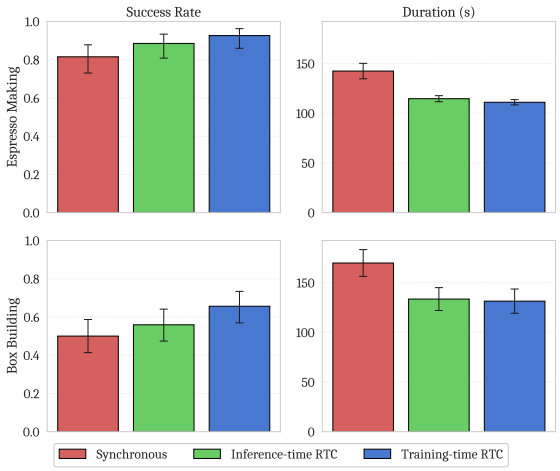


Figure 5.5: Real-world results: success rate and duration for espresso making and box building. Training-time and inference-time RTC perform similarly, while both improving speed over synchronous inference. Error bars represent 68% Wilson score intervals for success rate and ± 1 SEM for duration.

Part III

Policy Generalization via World Models

Chapter 6

SuSIE: Subgoal Synthesis via Image Editing

6.1 Introduction

A useful generalist robot must be able to — much like a person — recognize and reason about novel objects and scenarios it has never encountered before. For example, if a user instructs the robot to "hand me that jumbo orange crayon," it ought to be able to do so even if it has never interacted with a jumbo orange crayon before. In other words, the robot needs to possess not only the physical capability to manipulate an object of that shape and size but also the semantic understanding to reason about an object outside of its training distribution. As much as robotic manipulation datasets have grown in recent years, it is unlikely that they will ever include every conceivable instance of objects and settings, any more so than the life experiences of a person ever include physical interactions with every type of object. While these datasets contain more than enough examples of manipulating elongated cylindrical objects, they lack the broad semantic knowledge necessary to ground the *particular* objects that robots will encounter during everyday operation.

How can we imbue this semantic knowledge into language-guided robotic control? One approach would be to utilize models pretrained on vision and language to initialize different components of the robotic learning pipeline. Recent efforts, for example, initialize robotic policies with pretrained vision-language encoders [46] or apply pretrained models to semantic scene augmentation [59, 363]. While these methods bring semantic knowledge into robot learning, it remains unclear if these approaches realize the full potential of Internet pretraining in improving robotic policy execution at every level, or whether they simply improve the high-level visual generalization of the policy.

In this chapter, we propose an approach for leveraging pretrained image-editing models to enable generalizable robotic manipulation. We finetune the image-editing model on video

data such that, given the current frame and a language description of the current task, the model generates a hypothetical *future* frame. This does not require the model to precisely understand the intricacies of the robot's low-level dynamics and therefore should facilitate transfer from other data sources (e.g., human videos) where the low-level physical interactions and precise embodiment do not match. At test time, we employ a low-level goal-reaching policy trained on robot data to reach this hypothetical future frame; this policy, in turn, only needs to infer visuo-motor relationships to determine the correct actuation and does not need to understand the overlying semantics. Furthermore, such subgoals can simplify the task by inferring likely poses for the arm at intermediate substeps, such as the pose corresponding to grasping an object (see Figure 6.1). In fact, we observe in our experiments that even when existing approaches possess sufficient semantic understanding to solve a task, they often fail due to imprecise localization of obstacles and objects; following the generated subgoals enables our method to perform well in such scenarios. Much like a person first constructs a high-level plan to complete a task before deferring to muscle memory for the low-level control, our method can also be viewed as first running a high-level planner with integrated semantic reasoning *and* visual understanding before deferring to a low-level controller to execute the plan.

The main contribution of our work is **SU**bgoal **S**ynthesis via **I**mage **E**diting (**SuSIE**), a simple and scalable method for incorporating semantic information from pretrained models to improve robotic control. A pretrained image-editing model is used with minimal modification, requiring only finetuning on video data. The low-level goal-conditioned policy is trained with standard supervised learning, and faces the comparatively easy problem of reaching nearby image subgoals; this typically only requires attending to a single object or the arm position, ignoring most parts of the scene. Together, this approach achieves state-of-the-art results on the CALVIN benchmark and solves real robot control tasks involving novel objects, novel distractors, and even novel scenes. It outperforms all prior approaches on these real-world tasks, including oracle baselines with privileged information and RT-2-X [66] — a 55 billion parameter model trained on Internet-scale vision-language data as well as an order of magnitude more robot data than SuSIE.

6.2 Related Work

Incorporating semantic information from pretrained vision-language models. Prior works that incorporate semantic information from pretrained vision-language models (VLMs) into robot learning can be classified into two categories. The first category aims to improve visual scene understanding in robot policies with information from VLMs. For instance, GenAug [59], ROSIE [363], DALL-E-Bot [150], and CACTI [213] use text-to-image generative models to produce semantic augmentations of a given scene with novel objects and arrangements and train the robot policy on the augmented data to enable it to perform well in a similar scene. MOO [306] uses a pretrained object detector to extract bounding boxes to guide the policy towards the target object. Other works directly train language and

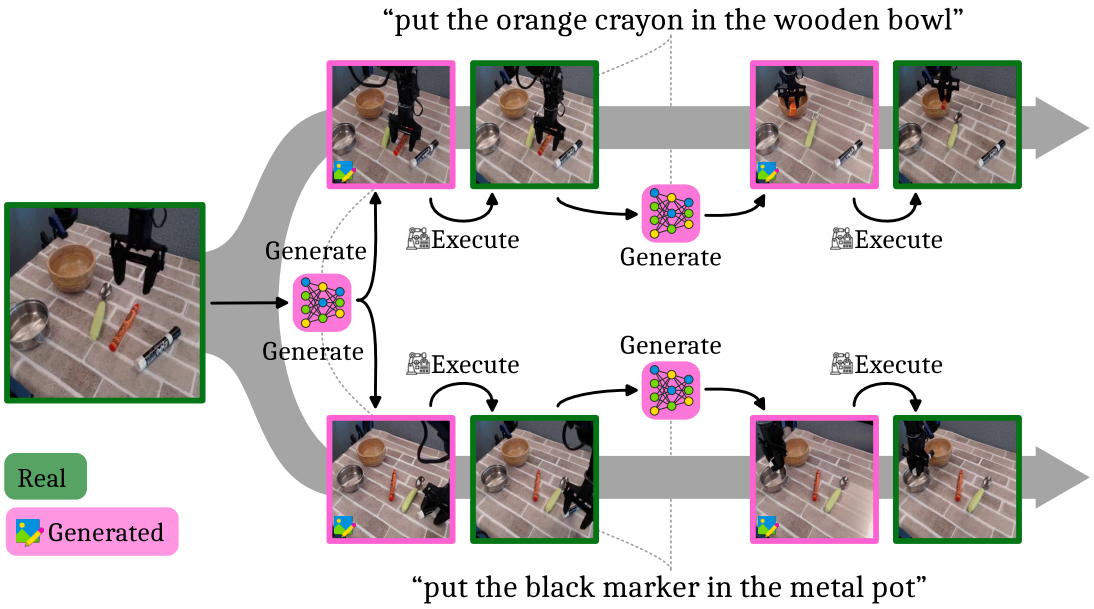


Figure 6.1: **(SuSIE)** Our method leverages a pretrained image-editing model to generate future subgoals based on language commands. A low-level goal-reaching policy then executes the actions needed to reach each subgoal. Alternating this loop enables us to solve the task.

image-based policies [42, 46, 294], some using VLM initializations [79, 264], to produce action sequences.

While these approaches do utilize pretrained models, we find in our experiments that pretraining using VLMs [46] does not necessarily enhance low-level policy execution in the sense that learned policies often localize the object or move the gripper imprecisely (see Figure 6.3). On the other hand, our approach is able to incorporate benefits of pretraining into low-level control by synthesizing subgoals that carefully steer the motion of the low goal reaching policy, improving its precision. While our approach can be directly applied in *unstructured* open-world settings, applying methods from [59, 363, 213] requires additional prior information, such as object bounding boxes or 3D meshes. This significantly restricts their applicability to scenarios where this additional information is not available: for example, GenAug [59] is not applicable in our experiments since 3D object meshes for new target objects were not available. Distinct from our approach for utilizing generative models, other works design representation learning objectives for vision-language pretraining for control [235, 209, 153, 25], but these methods still need to utilize limited amounts of data from the target task to learn a policy. In contrast, our approach solves the target task in zero-shot, without any task-specific data.

The second category of approaches incorporates semantic information from pretrained models for informing low-level controllers. Typically, these prior methods use pretrained models to

imagine visual [82, 6] or textual plans [41, 132, 131, 184], which then inform a low-level robot control policy. Of these, methods that train a low-level policies conditioned on text suffer from a grounding problem, which our approach circumvents entirely since the low-level policy only sees images.

Perhaps the most related methods to our approach in this second category are UniPi [82] and HiP [6], which train video models to generate a sequence of frames achieving the target task, and then extract robot actions by training an inverse dynamics model. Our approach does *not* attempt to generate full videos (i.e., *all* frames in a rollout), but only the next waypoint that a low-level policy must achieve to solve the commanded task. While this difference might appear small, it has major implications: modeling an entire video puts a very high burden on the generative model, requiring the frames to obey strict physical consistency. Unfortunately, we find that open-source video models often produce temporally inconsistent frames, which confuse the low-level controller (see Appendix E.3). Our method provides more freedom to the low-level controller to handle the physical aspects of the task over a longer time interval while guiding it at a level that is suitable to the diffusion model's ability to preserve physical plausibility. In our experiments, we find that our method significantly improves over an open-source reimplementation of UniPi [82].

Classical model-based RL and planning with no pretraining. The idea behind our approach is also related to several methods in the deep RL literature that do not use pretrained models and generally do not study language-guided control. For instance, [111, 170, 361, 340, 266, 112] train action-conditioned dynamics models and run RL in the model. While our approach also models multi-step dynamics, our model is not conditioned on an action input. Removing the dependency on an action input enables us to de-couple the finetuning of the (large) image-editing model from the policy entirely, improving simplicity and time efficiency. APV [282] trains an action agnostic dynamics model from videos but finetunes it in a loop with the policy, and hence, does not enjoy the above benefits. Finally, these model-based RL methods do not exhibit zero-shot generalization abilities to new tasks, which is an important capability that our method enjoys. Our approach is also related to several video prediction methods [87, 171, 13, 327] but utilizes a better neural network architecture (i.e., diffusion models instead of LSTMs and CNNs). Most related is to our method is hierarchical visual foresight (HVF) [234]: while HVF uses MPC to find an action, our approach uses a goal-reaching policy thereby eliminating the cost of running MPC with large dynamics models.

Our approach is also related to prior works that use generative models for planning in a single-task setting, with no pretraining. Trajectory transformer (TT) [138], decision transformer (DT) [56], and their extensions condition the policy on the target return or goal. While diffusion-based variants of these methods [139, 7] use diffusion models like our approach, they still require training data from the target task to learn a policy, unlike our zero-shot planning approach.

6.3 Preliminaries and Problem Statement

We consider the problem setting of language-conditioned robotic control. Specifically, we want a robot to accomplish a task described by a novel language command. We divide sources of data into 3 categories: language-labeled video clips $\mathcal{D}_l$ that do not include robot actions; language-labeled robot data $\mathcal{D}_{l,a}$ that include both language labels and robot actions; and unlabeled robot data $\mathcal{D}_a$ that include only actions (e.g., play data).

Formally, we define $\mathcal{D}_{l,a} = \{(\tau^1, l^1), (\tau^2, l^2), \cdots, (\tau^N, l^N)\}$, where each trajectory τ^n consists of a sequence of images (or states) $\mathbf{s}_i^n \in \mathcal{S}$ and actions $\mathbf{a}_i^n \in \mathcal{A}$ that were executed while collecting this data, i.e. $\tau^n = (\mathbf{s}_0^n, \mathbf{a}_0^n, \mathbf{s}_1^n, \mathbf{a}_2^n, \cdots)$, following the standard assumptions of a Markov decision process. l^n is a natural language command describing the task accomplished in the trajectory. $\mathcal{D}_l$ and $\mathcal{D}_a$ are organized similarly, but are missing either actions $\mathbf{a}_i^n$ or language annotations l^n, respectively. At test time, given a new scene $\mathbf{s}_0^{\text{test}}$ and a new natural language description l^{test} of a task, we evaluate a method in terms of its success rate at accomplishing this task starting from $\mathbf{s}_0^{\text{test}}$.

6.4 SuSIE: Subgoal Synthesis via Image Editing

Our goal is to utilize semantic information from the Internet to improve language-guided robot control in the presence of novel environments, scenes, and objects. How can we do this when models trained on general-purpose Internet data do not provide guidance in selecting low-level actions? Our key insight is that we can effectively leverage the capabilities of the pretrained model if we decouple the robot control problem into two phases: **(I)** generating subgoals that would need to be attained to succeed at the task, and **(II)** learning low-level control policies for reaching these generated subgoals. Our method incorporates semantic information from both text-image pretraining on Internet data as well as non-robot video data in Phase (I) by finetuning a text-guided image-editing model on $\mathcal{D}_l \cup \mathcal{D}_{l,a}$. Phase (II) is accomplished via a goal-conditioned policy trained on the robot data $\mathcal{D}_{l,a} \cup \mathcal{D}_a$. We describe each of these phases below and then summarize the resulting algorithm.

Phase (I): Synthesizing Subgoals From Image-Editing Models

The primary component of our method is a generative model that, given a target task specified by natural language, can guide a low-level controller to a state that advances the task. One way to accomplish this is to train a generative model to produce an immediate next subgoal image. We can then incorporate semantic information from the Internet into our algorithm by initializing this generative model with a suitable pretrained initialization, followed by finetuning it on multi-task, diverse video data consisting of robot rollouts as well as other videos from the Internet.

What is a good pretrained initialization for this model? Our intuition is that, since accomplishing a task is equivalent to "editing" the pixels of an image of the robot workspace under

constraints prescribed by the language command, a good pretrained initialization may be provided by a text-guided image-editing model. We instantiate our approach with Instruct Pix2Pix [47], though other image-editing models could also be used. Formally, this model is given by $p_\theta(\mathbf{s}_{\text{edited}} \mid \mathbf{s}_{\text{orig}}, l)$. Then, using the dataset $\mathcal{D}_l \cup \mathcal{D}_{l,a}$ of language-labeled video clips and robot trajectories, we finetune p_θ to produce valid subgoals $\mathbf{s}_{\text{edited}}$ given an initial image $\mathbf{s}_{\text{orig}}$ and a language label l. Formally, the training objective is given by

$$\max_\theta \quad \mathbb{E}_{\ (\tau^n, l^n) \sim \mathcal{D}_l \cup \mathcal{D}_{l,a};\ \ \mathbf{s}_i^n \sim \tau^n;\ \ j \sim q(j|i)} \left[\log p_\theta \left(\mathbf{s}_j^n \mid \mathbf{s}_i^n, l^n \right) \right], \tag{6.1}$$

where $q(j \mid i)$ is a distribution of our choosing that controls what subgoals the model is trained to produce. We want the diffusion model to generate subgoals that are a consistent distance in the future from the current state — close enough to be reachable by the low-level policy, but far enough to achieve significant progress on the task. As such, we choose dataset-dependent hyperparameters $k_{\min}$ and $k_{\max}$, and set q to select subgoals uniformly between $k_{\min}$ and $k_{\max}$ steps in the future, i.e.,

$$q(j \mid i) = U\big(j;\ [i + k_{\min}, i + k_{\max})\big).$$

Phase (II): Reaching Generated Subgoals with Goal-Conditioned Policies

In order to utilize the finetuned image-editing model to control the robot, we additionally need to train a low-level controller to select suitable robot actions. In this section, we present the design of our low-level controller, followed by a full description of our test-time control procedure. Since the image-editing model in SuSIE produces images of future subgoals conditioned on natural language task descriptions, our low-level controller can simply be a language-agnostic goal-reaching policy.

Training a goal-reaching policy. Our goal-reaching policy is parameterized as $\pi_\phi(\mathbf{a} \mid \mathbf{s}_i, \mathbf{s}_j)$, where $\mathbf{s}_j$ is a future frame that the policy intends to reach by acting at $\mathbf{s}_i$. At test time, we only need the low-level policy to be proficient at reaching close-by states that lie within $k_{\max}$ steps of the current state, since the image-editing model from Phase (I) is trained to produce subgoals within $k_{\max}$ steps of any state. To train this policy, we run goal-conditioned behavioral cloning (GCBC) on the robot data $\mathcal{D}_{l,a} \cup \mathcal{D}_a$. Formally, the training objective is given by

$$\max_\phi \quad \mathbb{E}_{\ \tau^n \sim \mathcal{D}_{l,a} \cup \mathcal{D}_a;\ (\mathbf{s}_i^n, \mathbf{a}_i^n) \sim \tau^n;\ j \sim U([i, i + k_{\max} + k_\delta))} \left[\log \pi_\phi(\mathbf{a}_i^n \mid \mathbf{s}_i^n, \mathbf{s}_j^n) \right], \tag{6.2}$$

where k_δ is another hyperparameter that provides a small amount of overhead, since the image-editing model is not perfect and may not always produce subgoals that are reachable in less than $k_{\max}$ steps, especially for unseen tasks.

Test-time control with π_ϕ and p_θ. Once both the goal-reaching policy π_ϕ and the subgoal generation model p_θ are trained, we utilize them together to solve new manipulation tasks based on user-specified natural language commands. Given a new scene $\mathbf{s}_0^{\text{test}}$ and a language command l^{test}, SuSIE attempts to solve the task by iteratively generating subgoals and commanding the low-level policy to reach these subgoals. At the start, we sample the first subgoal $\widehat{\mathbf{s}}_+ \sim p_\theta(\mathbf{s}_+ \mid \mathbf{s}_0^{\text{test}}, l^{\text{test}})$. Once the subgoal is generated, we then roll out the goal-reaching policy π_ϕ conditioned on $\widehat{\mathbf{s}}_+$ for k_{test} timesteps where k_{test} is a test-time hyperparameter. After k_{test} timesteps, we refresh the subgoal by sampling from the subgoal generation model again and repeat the process. In practice, for computational efficiency, we set k_{test} to be similar to the corresponding $k_{\max}$ used with the robot data and found this to be sufficient for obtaining good performance. However, given an unlimited computational budget, conventional wisdom would suggest that regenerating subgoals more often would lead to more robust control. Pseudocode for test-time control is provided in Algorithm 2.

Algorithm 2 SuSIE: Zero-Shot, Test-Time Execution

Require: Subgoal model $p_\theta(\mathbf{s}_+ \mid \mathbf{s}_t, l)$, policy $\pi_\phi(\mathbf{a} \mid \mathbf{s}_t, \mathbf{s}_+)$, time limit T, subgoal sampling interval k_{test}, initial state $\mathbf{s}_0^{\text{test}}$, language command l^{test}

$t \leftarrow 0$
while $t \leq T$ **do**
  Sample $\widehat{\mathbf{s}}_+ \sim p_\theta(\mathbf{s}_+ \mid \mathbf{s}_t^{\text{test}}, l^{\text{test}})$ ▷ Generate a new subgoal every k_{test} steps
  for $j = 1$ to k_{test} **do**
    Sample $\mathbf{a}_t \sim \pi_\phi(\mathbf{a} \mid \mathbf{s}_t^{\text{test}}, \widehat{\mathbf{s}}_+)$ ▷ Predict the action from current state and subgoal
    Execute $\mathbf{a}_t$
    $\mathbf{s}_{t+1}^{\text{test}} \leftarrow$ robot observation
    $t \leftarrow t + 1$

Implementation Details

In Phase (I), we utilize the pretrained initialization from the InstructPix2Pix model [47]. We implement Equation 6.1 using the standard variational lower bound objective for training diffusion models [120]. Our diffusion model and policy operate on images of size 256×256. To ensure that this model pays attention to the input image and the language command, we apply classifier-free guidance [122] separately to both the language and the image, following InstructPix2Pix. To obtain a robust goal-reaching policy in Phase (II), we use a diffusion policy [62, 114] that predicts chunks of 4 actions and performs temporal averaging over these predictions at test time [375]. More details about the training hyperparameters and architecture are provided in Appendix E.1.

6.5 Experimental Evaluation

The goal of our experiments is to evaluate the efficacy of SuSIE at improving generalization and the low-level policy execution in open-world robotic manipulation tasks. To this end, our experiments aim to study the following questions:

1. Can SuSIE solve a task in a novel environment, with novel objects, and with a novel language command, in zero-shot?
2. Does SuSIE exhibit an elevated level of precision and dexterity compared to other approaches that do not use subgoals?
3. How crucial is pretraining on Internet data, as well as cotraining on non-robot video data, for attaining zero-shot generalization?

To answer these questions, our experiments compare SuSIE to several prior methods including state-of-the-art approaches for training language-conditioned policies that leverage pretrained vision-language models in a variety of ways.

Experimental Scenarios and Comparisons

Real-world experimental setup and datasets. We conduct our real-robot experiments on a WidowX250 robot platform. Our dataset is BridgeData V2 [329], a large and diverse dataset

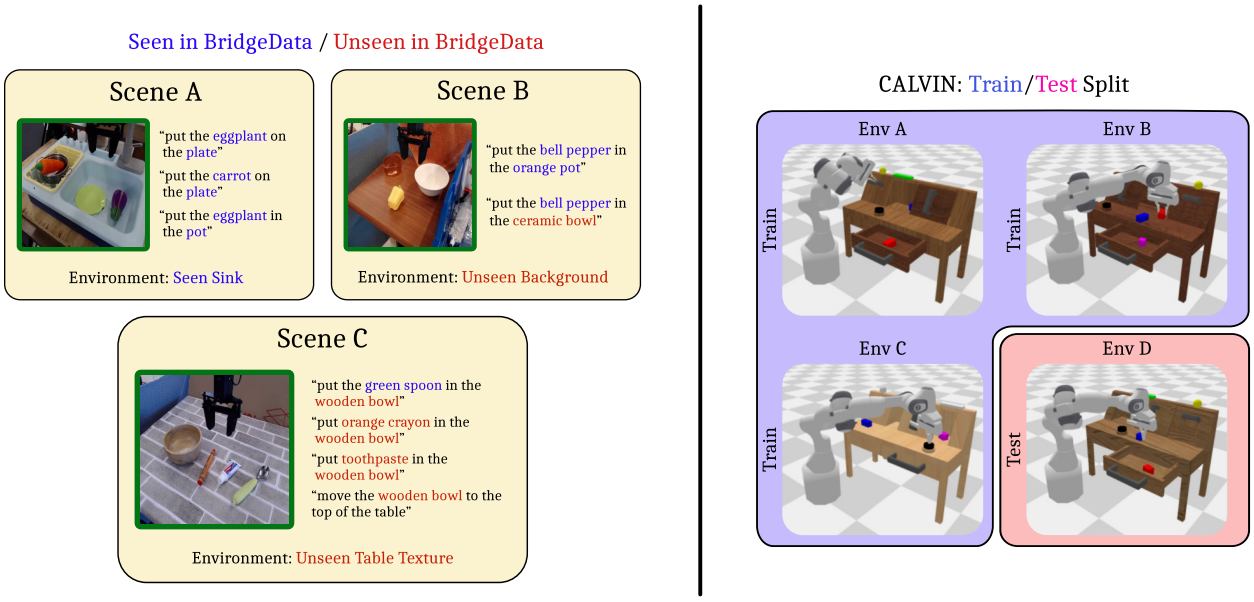


Figure 6.2: **(Experimental setup) (L)** We evaluate our method on 9 tasks across 3 real-world scenes. In terms of semantic generalization, the scenes become progressively more difficult, due to both an increasing visual departure from the robot training data and an increasingly confounding mixture of both seen and unseen objects. **(R)** In simulation, we evaluate our method in the zero-shot setting of the CALVIN benchmark, which involves training on 3 environments (A, B, and C) and testing on a 4th environment (D). The environments differ in table texture, positioning of furniture elements, and possible configurations of various colored blocks. Each environment comes with 34 language-specified tasks.

of robotic manipulation behaviors designed for evaluating open-vocabulary instructions. The dataset contains over 60k trajectories, 45k of which are language-labeled, which we use as our language-labeled robot dataset $\mathcal{D}_{l,a}$. We use the remaining 15k trajectories as our action-only dataset $\mathcal{D}_a$.

Our video-only dataset $\mathcal{D}_l$ is the Something-Something dataset [102], a dataset consisting of short video clips of humans manipulating various objects. We chose Something-Something because it primarily contains examples of object manipulation with a still camera frame and hence exhibits a smaller domain gap to the robot data collected with an over-the-shoulder camera compared to other video datasets that contain substantial egocentric motion. We filter the Something-Something dataset using its labels to remove trajectories that contain egocentric motion or otherwise lack significant manipulation behavior, producing a final dataset size of approximately 75k video clips.

Our evaluations present three different scenes (Figure 6.2) designed specifically to test the ability of various methods at different levels of open-world generalization: **Scene A:** this scene includes an environment and objects that are well-represented in BridgeData V2; **Scene B:** this scene is situated in an environment with a seen tabletop but a novel background and distractors, where the robot must move a seen object (bell pepper) into a choice of seen container (orange pot) or unseen container (ceramic bowl); and **Scene C:** this scene includes a table texture unlike anything in BridgeData V2 and requires manipulating both seen and unseen objects. Semantically, Scene C is the most challenging since the robot needs to carefully ground the language command to identify the correct object while resisting its affinity for an object that is well-represented in the robot data (the spoon). Scene B also requires semantic grounding to distinguish between the seen and unseen receptacles, while adding the additional challenge of manipulating the plastic bell pepper — this object requires particular precision to grasp due to being light, smooth, and almost as wide as the gripper.

Simulation tasks. We run our simulation experiments in CALVIN [226], a benchmark for long horizon, language-conditioned manipulation. CALVIN consists of four simulated environments, labeled A through D, and each environment comes with a dataset of human-collected play trajectories. Approximately 35% of the play data is annotated with language, which we use as our language-labeled robot dataset $\mathcal{D}_{l,a}$. We use the remaining 65% of the play data as our action-only robot dataset $\mathcal{D}_a$, and do not use any video-only dataset $\mathcal{D}_l$ in our simulation experiments.

Each environment consists of a Franka Emika Panda robot arm positioned next to a desk with various manipulatable objects, including a drawer, sliding cabinet, light switch, and various colored blocks. Environments are differentiated by their table textures, positions of the furniture objects, and configurations of colored blocks. With this benchmark, we study the most challenging zero-shot multi-environment scenario: training on A, B, and C, and testing on D. We follow the evaluation protocol from Mees et al. [226]. During evaluation, a

policy is given 360 timesteps to complete a chain of five language instructions. We provide more details about our experimental setup in Appendix E.2.

Comparisons. Our experiments cover methods that utilize pretrained models of vision and language in language-guided robot control in a variety of ways. Several prior methods tackle language-based robotic control as discussed in Section 6.2. In our experiments, we choose to compare to a representative subset of these prior methods to maximally cover the possible set of comparisons. In our real-world experiments, we compare to the following methods:

- **RT-2-X** [66], a 55 billion parameter vision-language model finetuned on the full Open X-Embodiment dataset to produce robot actions. The robot training data includes all of BridgeData V2, as well as a vast quantity of additional robot manipulation data totaling over 1.1 million trajectories.
- **MOO** [306], which utilizes pretrained object detectors to obtain 2D localization information for the manipulation targets and trains a language-conditioned behavioral cloning (LCBC) policy with this extra information. We re-implement MOO by using a large language model to extract the manipulation targets from the unstructured language labels in BridgeData V2, then OWL-ViT [229] to extract 2D bounding boxes for these objects. Unlike MOO, at test time we manually give the policy the ground truth 2D bounding boxes of the manipulation targets.
- **UniPi** [82], which finetunes a language-conditioned video prediction model on robot data. Since the original UniPi model utilized proprietary pretrained initializations that are not available publicly, we replicated this method in two different ways: (1) using the UniPi model from HiP [6], which is a PVDM [360] latent video diffusion model, and (2) by implementing our own image-space video diffusion model with a factorized spatial-temporal 3D UNet similar to Ho et al. [125]. We found that (1) was not able to produce reasonable-looking videos in the real robot scene (Appendix E.3), so we only evaluated (2) in our real-world experiments.
- **LCBC**, which trains a low-level policy conditioned on the language labels in BridgeData V2. This method is broadly representative of prior language-conditioned behavioral cloning methods such as RT-1 [42] and BC-Z [136]. We use the same architecture and hyperparameters as the low-level policy in SuSIE, as we found that they outperformed RT-1 on the BridgeData tasks.

We provide more details about the baselines and their implementations in Appendix E.1.

In our simulation experiments, we compare to additional methods previously studied on CALVIN. This includes methods that explicitly tackle long-horizon language-based control

on CALVIN such as multi-context imitation (MCIL) [207], hierarchical universal language-conditioned policy (HULC) [224], and improved variants of HULC [99]. We also compare to other state-of-the-art methods from Ge et al. [99] that employ an identical training and evaluation protocol as our experiments, namely MdetrLC [149] and AugLC [247].

Can SuSIE Perform Zero-Shot Robotic Manipulation?

Simulation results. We present performance in Table 6.1, in terms of success rates (out of 1.0) for completing chains of language instructions. SuSIE obtains superior zero-shot performance (train A, B, C $\rightarrow$ test D) compared to the previous state-of-the-art. We find that our reimplementation of UniPi achieves nontrivial success, but is still significantly outperformed by SuSIE.

Real-world results. We present performance of real-world evaluations in Table 6.2. SuSIE achieves the best performance across the board, beating RT-2-X, a 55 billion parameter model trained on significantly more robot and Internet data. As expected, all methods perform well in Scene A, which is well-represented in the robot data. SuSIE performs uniquely well in Scene B, as it is the only method that can consistently grasp the bell pepper.

In Scene C, while SuSIE still achieves the best performance, RT-2-X comes in a close second. We hypothesize that this is because — in contrast to Scene B — all of the objects in Scene C are easy to grasp. Therefore, the low-level precision of the policy is less important, which is the primary weakness of RT-2-X, as we discuss further in the next section. Qualitatively, we observed that the failure cases in Scene C for both SuSIE and RT-2-X were almost always imprecise manipulations (failed grasps or early dropping) rather than semantic misunderstanding; of the 4 objects, the toothpaste is the most difficult to grasp, which is why its success rate is the lowest. That is, both methods solve the semantic understanding component of the tasks, but SuSIE's improved low-level precision allows it to perform slightly better.

The performance of LCBC is as expected: since it is trained only on BridgeData V2, it has no way of grounding novel objects, and often puts the bell pepper in the wrong receptacle (in Scene B) or attempts to grasp the wrong object (in Scene C). Surprisingly, it can recognize the wooden bowl in Scene C, even though that exact object does not appear in the training data; we hypothesize that the wooden bowl is distinctive enough to recognize from the word "bowl" alone combined with the many other bowls in BridgeData V2. MOO underperforms expectations, even with its privileged test-time information in the form of ground-truth bounding boxes. We hypothesize that, despite our best attempts, the target objects and bounding boxes extracted at train time are noisy due to the unstructured nature of BridgeData V2 and therefore only serve to confuse the policy (see Appendix E.1). UniPi's poor performance is due to the video model often producing temporally inconsistent frames (see Appendix E.3).

	Success Rate for No. of Instructions in a Row					Avg. No. of Instructions Completed
	1	2	3	4	5	
HULC [224]	0.43	0.14	0.04	0.01	0.00	0.62
MCIL [207]	0.20	0.00	0.00	0.00	0.00	0.20
MdetrLC [99]	0.69	0.38	0.20	0.07	0.04	1.38
AugLC [99]	0.69	0.43	0.22	0.09	0.05	1.43
LCBC	0.67	0.31	0.17	0.10	0.06	1.31
UniPi (HiP) [6]	0.08	0.04	0.00	0.00	0.00	0.12
UniPi (Ours) [82]	0.56	0.16	0.08	0.08	0.04	0.92
SuSIE (Ours)	**0.87**	**0.69**	**0.49**	**0.38**	**0.26**	**2.69**

Table 6.1: **(CALVIN benchmark performance)** SuSIE is able to chain together more instructions with a higher success rate than all prior methods in the zero-shot (train A, B, C $\rightarrow$ test D) setting. We provide an example rollout in Appendix E.4.

Does SuSIE Improve Precision and Low-Level Skill Execution?

Our real-world and simulated results clearly demonstrate the efficacy of SuSIE in executing novel language commands in a variety of scenes. In Section 6.5, we hypothesize that the advantage of SuSIE is twofold: semantic generalization due to Internet and video pretraining, and low-level precision due to subgoal guidance. In this section, we aim to validate the latter half of this hypothesis. To this end, we train an **Oracle GCBC** policy, which is identical to the low-level policy in SuSIE except that it is trained with a goal horizon that extends to the end of each trajectory. At test time, we provide the policy with a privileged *real goal image* of the fully completed task, eliding the need for any semantic understanding. Thus, the only advantage of SuSIE over Oracle GCBC is the subgoal guidance, while SuSIE is disadvantaged by additionally needing to interpret the language instruction and generate subgoals for the correct task.

As demonstrated in Table 6.3, although Oracle GCBC is a strong baseline, SuSIE is still the best-performing method on average. In particular, we observed that Oracle GCBC is still unable to grasp the bell pepper in Scene B due to a lack of low-level precision. Figure 6.3 provides a qualitative visualization of SuSIE grasping the bell pepper while Oracle GCBC and RT-2-X fail to do so. These results validate the hypothesis that, in addition to enabling semantic generalization, SuSIE also improves low-level precision and dexterity by decomposing the problem into a two-level hierarchy.

	Task	LCBC	MOO	UniPi	RT-2-X	SuSIE (Ours)
Scene A	Eggplant on plate	0.9	0.4	0.0	0.3	**1.0**
	Carrot on plate	0.4	0.3	0.0	0.6	**0.9**
	Eggplant in pot	0.6	**0.7**	0.0	0.4	**0.7**
	Average	0.63	0.47	0.0	0.43	**0.87**
Scene B	Bell pepper in pot	0.1	0.0	0.0	0.0	**0.5**
	Bell pepper in bowl	0.3	0.1	0.1	0.0	**0.5**
	Average	0.20	0.05	0.05	0.00	**0.50**
Scene C	Toothpaste in bowl	0.0	0.0	0.0	0.5	**0.6**
	Crayon in bowl	0.0	0.0	0.0	0.9	**1.0**
	Spoon in bowl	0.1	0.3	0.1	0.7	**0.9**
	Bowl to top	0.6	0.1	0.1	0.9	**1.0**
	Average	0.18	0.10	0.05	0.75	**0.88**

Table 6.2: **(Real-world performance)** SuSIE achieves the best success rate across the board, demonstrating both high precision as well as the ability to generalize to novel environments, objects, and language commands.

Are Internet and Video Training Crucial for Zero-Shot Generalization?

Finally, we conduct an experiment to understand the effect of both Internet-scale pretraining (in the form of InstructPix2Pix initialization) and video cotraining (in the form of training on BridgeData V2 and Something-Something simultaneously). We train two additional subgoal generation models: one robot-only model trained with InstructPix2Pix initialization but only on BridgeData V2, and one from-scratch model trained without InstructPix2Pix initialization at all. The from-scratch model uses the same frozen text encoder, image encoder, and UNet architecture as InstructPix2Pix, but initializes the UNet with random weights.

We present sample generations from these models in Figure 6.4. The generations of the from-scratch model are consistently low-quality, either misunderstanding the task, producing significant artifacts, or failing to edit the image at all; this demonstrates the importance of initializing from Internet-scale pretrained weights. As one might expect, the BridgeData-only model is on par with the full Something-Something model on in-distribution tasks, which come from the BridgeData V2 validation set. However, on out-of-distribution tasks, the BridgeData-only model has an increased propensity to hallucinate background items or misunderstand the task. We found that for moving the wooden bowl, in particular, the full Something-Something model is the only one that can produce fully correct subgoals.

As additional evidence, we present a quantitative evaluation of the BridgeData-only model

	Task	Oracle GCBC	SuSIE (Ours)
Scene A	Eggplant on plate	0.7	**1.0**
	Carrot on plate	0.8	**0.9**
	Eggplant in pot	**1.0**	0.7
	Average	0.83	**0.87**
Scene B	Bell pepper in pot	0.1	**0.5**
	Bell pepper in bowl	0.0	**0.5**
	Average	0.05	**0.50**
Scene C	Toothpaste in bowl	**0.7**	0.6
	Crayon in bowl	**1.0**	**1.0**
	Spoon in bowl	0.8	**0.9**
	Bowl to top	0.6	**1.0**
	Average	0.78	**0.88**
CALVIN	Move slider right/left	0.52	**0.86**
	Turn on/off light bulb	0.74	**0.96**
	Open/close drawer	**1.00**	0.98
	Turn on/off LED	0.36	**1.00**
	Average	0.66	**0.95**

Table 6.3: **(Comparison to GCBC with oracle goals)** Executing generated subgoals improves performance even compared to executing privileged ground-truth final goals.

	Task	BridgeData Only	BridgeData + Something-Something
Scene B	Bell pepper in pot	0.2	**0.5**
	Bell pepper in bowl	0.4	**0.5**
	Average	0.30	**0.50**
Scene C	Toothpaste in bowl	**0.7**	0.6
	Crayon in bowl	0.9	**1.0**
	Spoon in bowl	**0.9**	**0.9**
	Bowl to top	0.7	**1.0**
	Average	0.80	**0.88**

Table 6.4: **(Comparison to BridgeData-only subgoal generation model)** Cotraining the subgoal generation model on the Something-Something dataset improves performance in Scenes B and C, which are unseen in BridgeData V2.

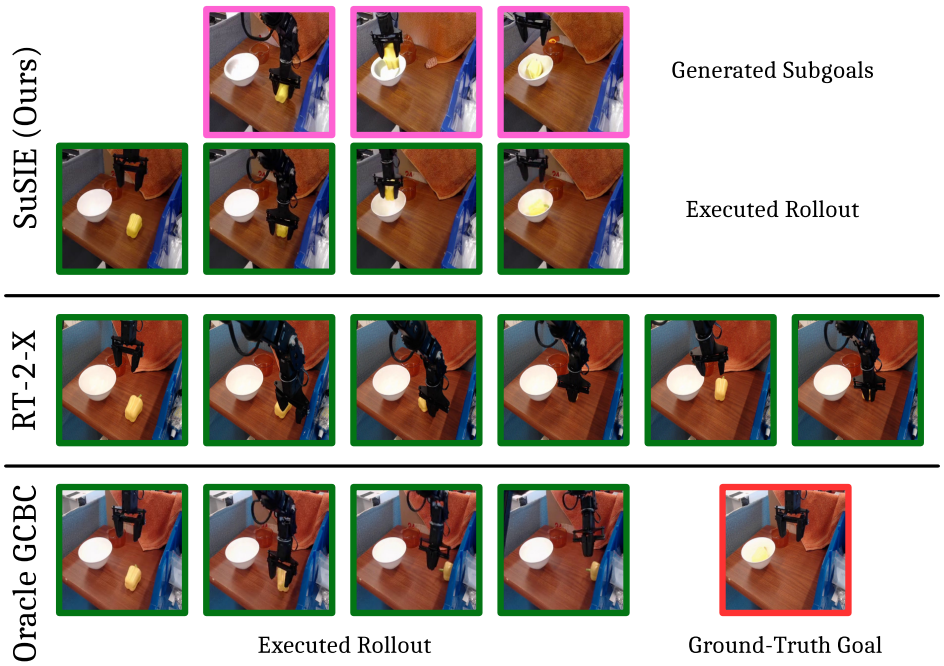


Figure 6.3: **(Visualizing rollouts)** Visualized rollouts from SuSIE, RT-2-X, and Oracle GCBC on the task "put the yellow bell pepper in the ceramic bowl" from Scene B. While RT-2-X and Oracle GCBC fail to grasp the object, the generated subgoals from SuSIE precisely guide the low-level controller, improving low-level skill execution for difficult manipulation tasks.

on the tasks from Scenes B and C in Table 6.4. In real-world task execution, the difference between the BridgeData-only model and the full Something-Something model is not as pronounced as in the qualitative samples. This is because we find that the low-level goal-conditioned policy is fairly robust to hallucinations of background items, suggesting that it primarily attends to the gripper pose and the target object. However, the overall increased quality of the subgoals from the full model still produces a small quantitative improvement on average.

6.6 Discussion

We presented a method for robotic control from language instructions that generates subgoals to guide a low-level goal-conditioned policy. The subgoals are generated by an image-editing diffusion model finetuned on video data. This system improves both zero-shot generalization to new objects and the overall precision of the policy because the subgoal model incorporates semantic benefits from pretraining and commands the low-level policy with fine-grained guidance. Our experiments show that SuSIE improves over prior techniques on the CALVIN benchmark and improves visual generalization and low-level control on



Figure 6.4: **(Comparison of subgoal quality)** A comparison between 3 subgoal generation models: one trained on both robot and video data (Bridge + Smthn-Smthn), used throughout this chapter; one trained on robot data only (Bridge Only); and one trained without InstructPix2Pix initialization (From Scratch). The top half (In Distribution) comes from the BridgeData V2 validation set, while the bottom half (Out of Distribution) comes from our evaluation Scenes B and C, unseen in BridgeData V2. Both Internet-scale pretraining and video cotraining and important for generating high-quality subgoals, especially in the zero-shot generalization setting.

real-world manipulation tasks. In the real world, SuSIE outperforms language-conditioned behavioral cloning, a privileged goal-conditioned policy that gets access to a ground-truth final goal, as well the state-of-the-art instruction-following approach, RT-2-X, which is trained on more than an order of magnitude more robot data (over 1.1 million trajectories for RT-2-X, vs. 60k for ours).

Our method is simple and provides good performance, but it does have limitations that suggest promising directions for future work. For instance, the diffusion model and the low-level policy are trained separately, meaning the diffusion model is unaware of the capabilities of the low-level policy — it is trained on the same data, but assumes that anything that is reachable in the data can also be reached by the policy. We hypothesize that performance can be improved by making the diffusion model aware of the low-level policy's capabilities. More broadly, we found the performance of our method to often be bottlenecked by the performance of the low-level policy, suggesting that addressing either of these limitations might lead to a more performant method for importing Internet-scale knowledge into robotic manipulation.

Chapter 7

$\pi_{0.7}$: a Steerable Generalist Robotic Foundation Model

7.1 Introduction

Foundation models work on the principle that generalist capabilities emerge from training on large and diverse datasets. For example, large language models can not only recall facts and semantic knowledge, but they can also compose that knowledge in new ways, solving problems that require unlikely connections, applying user-defined formats (e.g., JSON), and performing chain-of-thought reasoning. This kind of *compositional* generalization is arguably the cornerstone of generalist capabilities, but it has proven elusive in the domain of physical intelligence. While robotic foundation models such as vision-language-action models (VLAs) have advanced significantly in size and capability, their ability to generalize to new tasks or recombine skills in new ways has so far been limited. Unlike language models, which can compose different capabilities from their training data to solve new problems, prior VLAs not only lack the ability to solve new tasks, but often struggle to fluently perform all of the instructions they were trained on without task-specific fine-tuning.

In this paper, we present a new model, $\pi_{0.7}$, that exhibits strong signs of compositional generalization — enabling it to follow diverse language instructions, attain performance comparable to more specialized fine-tuned models on dexterous tasks, and even compose these behaviors in new ways. This is enabled by leveraging large and diverse datasets, including data from many robots with diverse strategies, suboptimal data from autonomous execution (including both data from RL post-trained agents as well as failures), and non-robot data from videos of humans performing tasks and general multimodal data from the internet. However, using such data naïvely does not lead to success: with a diversity of examples that differ in terms of both strategy and task performance, a naïve training process would lead to a model that averages together different modes in the dataset and produces suboptimal

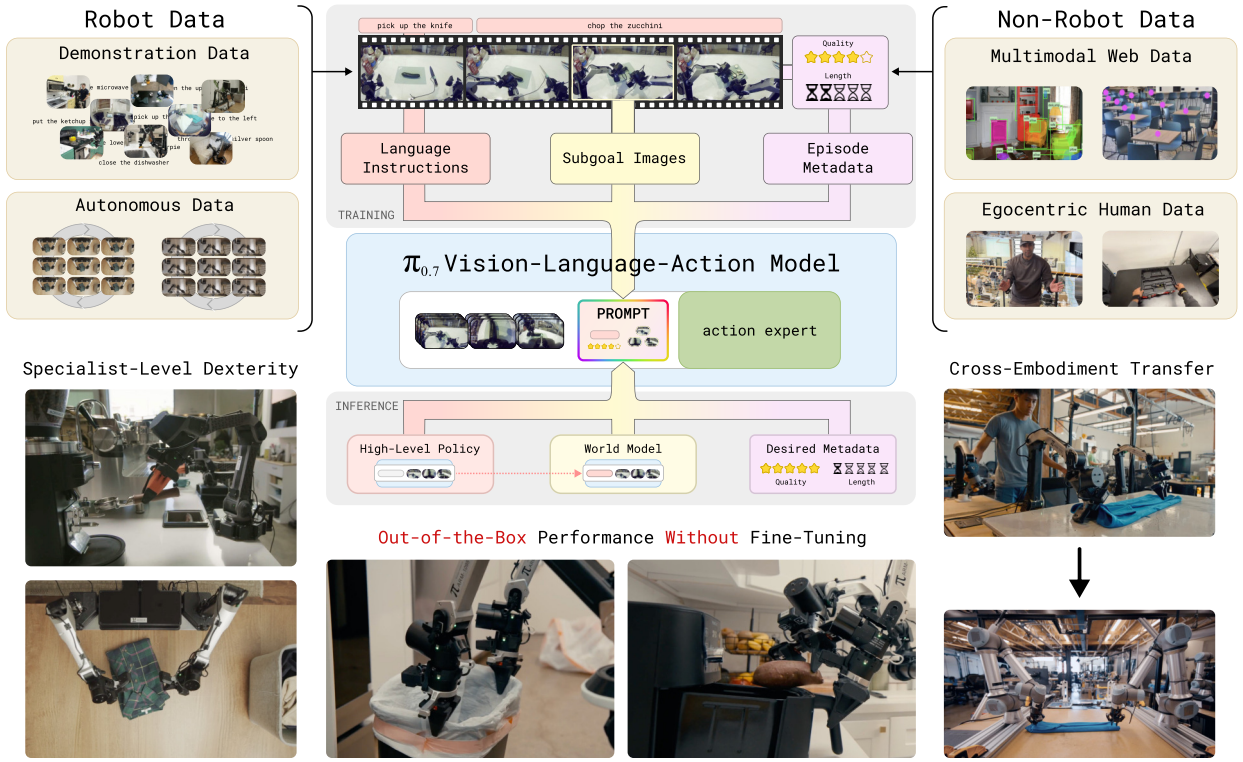


Figure 7.1: We introduce $\pi_{0.7}$, a steerable generalist robot foundation model that can perform dexterous tasks across many tasks, environments, and robots. $\pi_{0.7}$ is trained with diverse prompts that contain not only the task description, but detailed language, generated subgoal images, and episode metadata. This provides richer context about not only *what* to do, but also *how* to do it, making it possible for $\pi_{0.7}$ to leverage broad set of both robot and non-robot data and compose the skills in this data in new ways to solve new tasks.

results. In training $\pi_{0.7}$, we address this challenge by annotating the data with detailed context annotations that contain not only information about *what* to do but also *how* to do it and provide this knowledge to the model using a variety of multimodal conditioning signals. In this way, each episode teaches the robot about nuanced concepts and skills that it could use not only to perform the training tasks effectively, but also to compose in new ways to solve new tasks. Our proposed prompt structure includes detailed language labels, strategy metadata, and multimodal information such as subgoal images. This allows us to resolve the ambiguity in large and diverse datasets, learn from suboptimal behaviors without hurting performance, and obtain broad generalization across instructions, embodiments, and environments.

The idea that detailed prompts or context can improve the performance of foundation models

has been explored in other fields. For example, models for image and video generation utilize *prompt expansion* to produce high-quality generations. Our approach has many parallels to such methods. However, in robotics, simply *captioning* the data with more detailed text is not enough — the details that determine task success and proficiency might be more subtle (e.g., information about the overall quality of the episode), or might simply be hard to express with language alone (e.g., the particular appearance of a cleanly folded t-shirt). Therefore, in addition to using more detailed text, our model adds a range of additional metadata to the prompt, as shown in Fig. 7.1, including information about episode quality (strategy metadata), the control modality used by the robot, and subgoal images. Some of this information can be provided or omitted at test time, but including it in training results in a model that can more effectively compose the concepts that it was trained on and exhibit a variety of emergent capabilities.

In our evaluation, we show that $\pi_{0.7}$ exhibits a number of capabilities that go beyond prior robotic foundation models:

- **Out-of-the-box performance:** $\pi_{0.7}$ can reliably perform highly dexterous, long-horizon tasks such as using an espresso machine, folding laundry, taking out a trash bag, folding a box, and peeling vegetables, without any task-specific post-training and in a variety of environments.
- **Instruction generalization:** $\pi_{0.7}$ can follow a diverse set of language instructions in unseen environments and demonstrates robust generalization to complex, unseen language references. For example, $\pi_{0.7}$ can follow a diverse set of open-ended instructions in entirely unseen kitchen and bedroom environments.
- **Cross-embodiment generalization:** $\pi_{0.7}$ can enable zero-shot cross-embodiment transfer, making it possible to transfer dexterous tasks such as folding a t-shirt to a robot that was never trained to perform any laundry folding task, matching the performance of expert operators teleoperating the robot on their initial attempts.
- **Compositional task generalization:** $\pi_{0.7}$ can be instructed to perform new tasks by composing skills in previously unseen ways. For example, we can prompt $\pi_{0.7}$ to use new kitchen appliances, such as loading a sweet potato into an air fryer, or prompt it to perform tasks in new ways.

Through ablation and scaling studies, we also empirically demonstrate that there is a strong synergy between diverse datasets and detailed contexts: our approach enables learning from mixed-quality data and non-standard data sources without hurting the model performance, and diverse data boosts the model performance when detailed context information is provided during training.

7.2 Related Work

Generalist robot manipulation policies. There is a large body of work studying the development of generalist robot policies. These generalist policies are sometimes trained from scratch [42, 270, 315, 198, 331, 319], but are more commonly initialized using pre-trained vision-language models [43, 240, 161, 33, 338, 376, 311, 31, 377, 141, 179, 178, 260, 27, 365, 2, 382] or pre-trained video generation models [160, 245, 356, 339, 52]. Various works have developed architectural components of VLAs such as memory [378, 302, 291, 188, 92, 177, 371, 137, 322], hierarchy for long-horizon planning [3, 183, 293, 31, 311], and goal image conditioning [372]. We develop a VLA model that incorporates all three of these components in a single model, building on top of the $\pi_{0.6}$-MEM architecture [316, 322]. While generalist policies are most often trained on robot demonstration data, prior works have shown how to derive benefits from incorporating web data [43], egocentric videos of humans [354, 190, 155, 182, 352, 245, 204, 368], and autonomous robot experience [317, 347, 345] into pretraining. We incorporate all of these data sources and find that the combination of diverse data with detailed prompting yields a model with strong signs of compositional generalization and performant out-of-the-box behavior.

Generalization across tasks and embodiments. Much prior work has aimed to learn robot policies that generalize not only to different environments, objects, and backgrounds, but also to entirely new tasks and embodiments. Often, this is done by leveraging human video data, either for general representation learning [236, 210, 344, 26, 381, 20], by directly supervising with human motions [54, 288, 24, 14, 15, 154, 155, 292], or by extracting 2D point tracks [23, 326, 337, 105]. Other work has aimed to improve generalization by directly leveraging Internet pre-trained foundation models during training or inference [151, 214, 58, 362, 305, 76, 142]. With the increasing availability of large, cross-embodiment robot datasets [65], there has also been work on explicitly improving cross-embodiment transfer between robots [349, 74, 350, 366, 104, 4, 117]. Rather than leveraging existing datasets, some works have proposed specialized hand-held devices that can be used to collect data that can then generalize to various robot embodiments [63, 357]. In this work, we find that the right prompting allows our model to leverage diverse robot, human, and Internet data to achieve strong generalization across tasks and embodiments.

Prompting robots with subgoal images. A core architectural component of our model relative to $\pi_{0.6}$-MEM is to allow the model to be prompted using goal images, including generated subgoal images. Conditioning robot manipulation policies on goal images and videos has been explored in a large body of work. Some of these works utilize user-provided images [248, 53, 37, 230], while others condition the policy on generated goal images from a separate model [232, 233, 35, 164, 158, 185, 370, 84] or in a chain-of-thought fashion [372]. Alternatively, image and video generation can be integrated into policy training objectives [186, 160, 245, 364, 356, 83] to improve policy representations and produce more generalizable actions. We view our contribution as complementary to these works: we do not aim to

propose a new architecture or model design, so much as a methodology for enabling VLAs to utilize more diverse data sources, together with an empirical analysis showing that it leads to strong indications of compositional generalization. To our knowledge, our empirical results go significantly beyond the quantitative improvements reported in prior works, showing zero-shot transfer of dexterous skills like laundry folding to a different robot and generalization to novel object interactions such as operating an air fryer.

7.3 Flow-Based Vision-Language-Action Models

VLAs are trained by starting from a pre-trained vision-language model (VLM) backbone, and adapting it for robotic control. The training dataset $\mathcal{D}$ contains robot trajectories, which are sequences of observations $\mathbf{o}_t$ and actions $\mathbf{a}_t$. The observations $\mathbf{o}_t = [\mathbf{I}_t^1, \ldots, \mathbf{I}_t^n, \mathbf{q}_t]$ consist of n camera images $\mathbf{I}_t^i$ and the joint configuration of the robot $\mathbf{q}_t$, while the actions $\mathbf{a}_t$ consist of joint or end-effector commands.

VLAs are typically trained to predict an *action chunk*, corresponding to a short trajectory of future actions $\mathbf{a}_{t:t+H}$, based on a recent history of observations $\mathbf{o}_{t-T:t}$ (often a shorter horizon of actions, $\hat{H} < H$, is executed). The action chunk can be generated by an "action expert", a smaller transformer that attends to the VLM backbone and enables fast inference at runtime. The action expert typically uses a flow matching [192] (or diffusion) objective that captures the multi-modality of the robot actions. To learn effective representations, our model also uses the knowledge insulation (KI) training recipe [77]: the VLM backbone is supervised with FAST tokens [253], and while the action expert attends to all of the activations in the VLM backbone, gradients from the action expert do not flow into the VLM backbone, such that the VLM is trained via the comparatively stable discrete cross-entropy loss.

In addition to the observation and action, each training example for the VLA is accompanied by a *prompt* or *context*, which we denote with $\mathcal{C}_t$. Conventionally this corresponds to a language instruction ℓ_t, such that $\mathcal{C}_t = (\ell_t)$, provided by a human annotator (e.g., "clean up the kitchen").

In designing $\pi_{0.7}$, we explore how additional information added to the context for each training example can enable learning from diverse and heterogeneous datasets (including suboptimal behaviors and failures). As we will show, training with this data leads to a model with greater robustness and dexterity, and makes it possible for the model to generalize more broadly. The training objective for the VLA π_θ corresponds to an approximate log-likelihood given by

$$\max_\theta \; \mathbb{E}_{\mathcal{D}} \left[\log \pi_\theta(\mathbf{a}_{t:t+H} \mid \mathbf{o}_{t-T:t}, \mathcal{C}_t) \right]. \tag{7.1}$$

Note that a flow matching action expert optimizes an approximate lower bound rather than a closed form log-likelihood [33]. The dataset $\mathcal{D}$ typically consists of high-quality human demonstration trajectories; however, as mentioned, we use a broader dataset, which includes

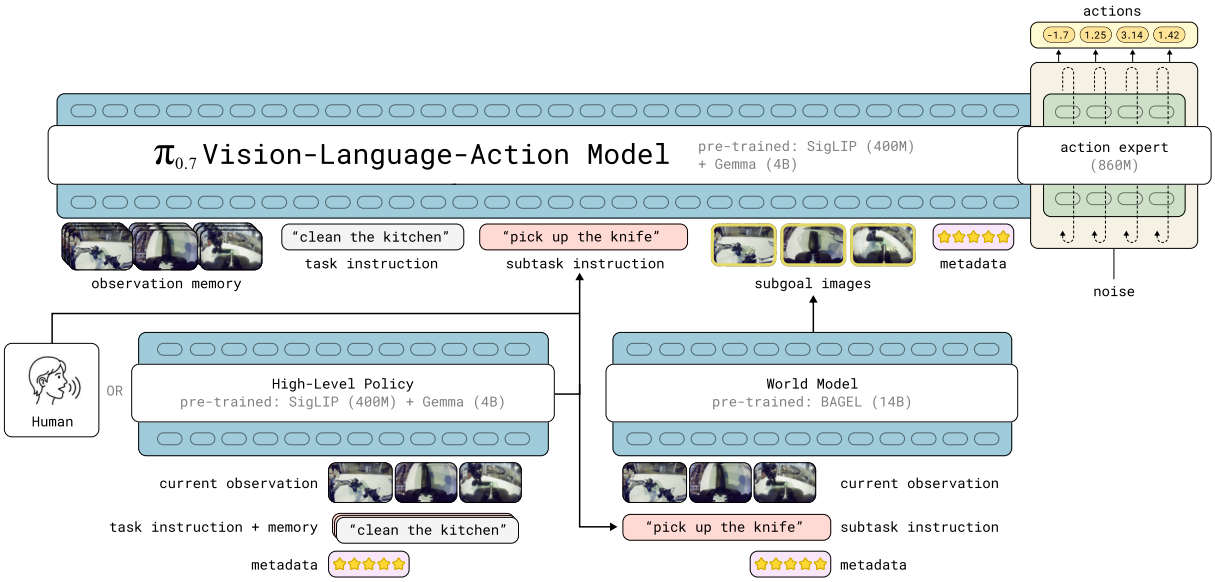


Figure 7.2: **Architecture overview.** The $\pi_{0.7}$ model is a 5B-parameter VLA consisting of a 4B VLM backbone, a MEM-style video history encoder and a 860M parameter action expert. The model's context includes multiple distinct modalities, including language commands, episode metadata that describes the data quality and strategy, and multimodal inputs such as subgoal images. At runtime, the language commands are produced by a high-level semantic policy based on the same architecture, and the subgoal images are produced by a lightweight world model based on the BAGEL image generation model [72].

failed episodes and suboptimal autonomous rollouts, as well as other data sources such as egocentric human video data. We will show how using a sufficiently detailed and informative context $\mathcal{C}_t$ makes it possible to incorporate such diverse data and, perhaps surprisingly, even results in better policy performance and generalization.

7.4 $\pi_{0.7}$ Overview

$\pi_{0.7}$ is our newest robotic foundation model that builds on the existing VLA architecture from $\pi_{0.6}$ [316] and the MEM memory system [322] and extends it with multi-modal context conditioning. The model consists of a VLM backbone initialized from the Gemma3 4B-parameter VLM [313] (including a 400M-parameter vision encoder), and a flow matching action expert with 860M parameters. The model has about 5B total parameters. The vision encoder is also initialized from Gemma3 and follows the design of the MEM video history encoder [322], applying both temporal and spatial compression over history observations and outputting a fixed number of tokens for any number of history frames. An overview of the model architecture is provided in Fig. 7.2, and Sec. 7.6 describes the architecture in more

detail.

Our previous models, π_0, $\pi_{0.5}$, and $\pi_{0.6}$, use a short textual description of the task as the context. In training $\pi_{0.7}$, we expand the context to include additional information and modalities: more expressive language commands, episode metadata, and subgoal images, making it possible to train on diverse and potentially suboptimal data.

7.5 Diversifying the Prompt

In this section, we describe each part of the prompt contained in the context $\mathcal{C}_t$ used by $\pi_{0.7}$. The model is trained to handle prompts that contain each of these components, though it is trained with each component randomly dropped out so that it can also handle any subset, providing flexibility at test time.

Subtask instructions

Following $\pi_{0.5}$ [31], we include intermediate, higher-level text that captures the **next semantic subtask** as part of the prompt in addition to the overall textual task description ℓ_t (e.g., "clean the kitchen"). We denote this intermediate text by $\hat{\ell}_t$ (e.g., "open the fridge door"). During inference, $\hat{\ell}_t$ may be produced by a learned high-level policy or a human (or be omitted) and may change over time. We collect data from a diverse set of tasks and scenarios, and then annotate the segments with detailed textual descriptions.

Conditioning the model on the semantic subtask also enables us to **verbally coach** the model step-by-step. Since the model is trained to follow diverse language instructions, it can follow the live instructions from the human in a new task, e.g., loading a sweet potato into an air fryer (Fig. 7.14). After coaching we can take the verbal coaching data to finetune $\pi_{0.7}$ as a high-level policy that maps the robot observations, task specification, and history of past subtask instructions to the new subtask instruction (Fig. 7.2 bottom left). This high-level policy then guides the robot to perform the task fully autonomously.

Subgoal images

While subtask instructions are effective at conveying the high-level intent of the task, they may lack details that matter for execution — e.g., "open the fridge door" does not specify how the robot arm should grasp the handle. Subgoal images address this by depicting the desired near-future state of the scene in images, providing a richer specification of *what the world should look like* after successful progress of the task.

We consider **multi-view subgoals** $\mathbf{g}_t = [G_t^1, \ldots, G_t^n]$, where G_t^i is the desired near-future image for camera i. Multi-view subgoals simultaneously specify environment- and object-

centric outcomes (often easiest in the base view) and arm/gripper outcomes (often easiest in wrist views), improving spatial grounding for control.

At runtime, the subgoal images are produced by a **lightweight world model**, which takes in the same subtask instruction $\hat{\ell}_t$ as the main model, but benefits from web-scale pre-training on videos and image editing tasks and is thus capable of generalizing to diverse tasks and scenarios. Generated subgoal images that are grounded in the robot's current observation can often more clearly disambiguate the objective for the policy than a language instruction, resulting in improvements in language following and generalization. We denote this model as g_ψ and it is trained with the objective

$$\max_{\psi} \mathbb{E}_{\mathcal{D}_g} \left[\mathcal{L}_{\text{CFM}} \left(\mathbf{g}_t^\star,\ g_\psi(\mathbf{o}_t, \hat{\ell}_t, m) \right) \right],$$

where $\mathcal{L}_{\text{CFM}}$ is the standard flow matching loss [192], $\mathbf{g}_t^\star$ is the future subgoal image, m is the episode metadata from Sec. 7.5, and the dataset $\mathcal{D}_g$ is a subset of segments from Sec. 7.5 that are annotated with especially high-quality subtask labels $\hat{\ell}_t$. The image frames at the end of the segments serve as the ground-truth subgoal, i.e., $\mathbf{g}_t^\star = \mathbf{o}_{t_{\text{end}}}$.

Following SuSIE [35], our world model is initialized using an off-the-shelf image generation and editing model with web-scale pre-training. We initialize from BAGEL [72], a 14B mixture-of-transformers [187] model capable of image understanding, editing, and generation. By augmenting our world model training with web data, non-robot data sources such as egocentric human videos, and other video data, we can acquire semantic and physical concepts from these other data sources and then transfer them into $\pi_{0.7}$ via subgoal images. Implementation details are in Appendix F.3.

Episode metadata

A key goal in expanding the context provided to the model is to train on a broader, more diverse dataset of trajectories. Instead of just using high-quality demonstration data, $\pi_{0.7}$ leverages lower quality demonstrations (including failures) and even autonomous data from prior models. Since we still want $\pi_{0.7}$ to perform the task as well as possible at test time, we need to appropriately label these diverse trajectories with information about *how* the task was performed so that the model can correctly contextualize them. To this end, we add a variety of "episode metadata" information to the context with attributes of the given training episode. We denote the set of metadata m, which may contain various labels including

- **Overall speed**: the length of the episode in timesteps. We discretize the values in an interval of 500 steps, i.e., values between 1750 and 2250 are binned to "2000 steps". Often faster speed also corresponds to higher quality, e.g., the episode has fewer mistakes.

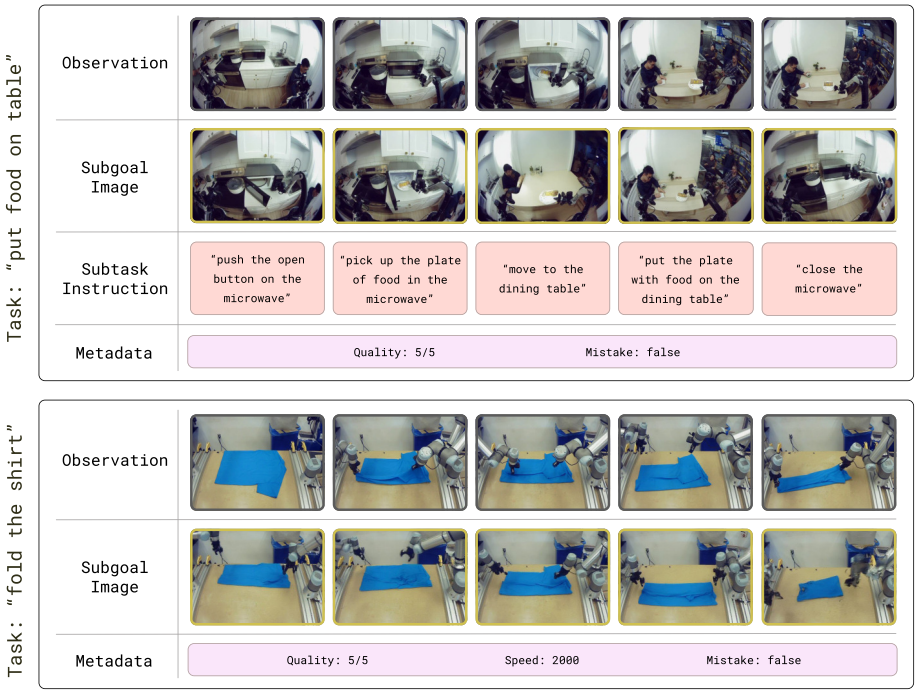


Figure 7.3: **Prompt overview.** $\pi_{0.7}$ uses diverse modalities of context in the prompt, including: subtask instructions, subgoal images, and episode metadata. We train the model with dropout for each component, and then prompt the model flexibly combining modalities. For example, when using the UR5e bimanual manipulator to fold a shirt, we use subgoal image and metadata prompting.

- **Overall quality**: task execution quality expressed as a score between 1 and 5, with 5 being the highest quality.
- **Mistake**: label indicating whether the robot made a mistake within a given action segment (e.g., failing to grasp an object or performing the wrong subtask). These labels are provided by humans coarsely annotating our data.

The $\pi_{0.7}$ model is thus trained with ground-truth episode speed and manual annotations of the episode quality and mistake segments from a diverse data mixture. The data diversity (e.g., episodes of varying speed) provides the necessary signals for the model to learn to correlate such metadata with the target action. At runtime, the model can then be instructed to perform the task at high speed, with high quality, and without mistakes, through metadata prompting.

Control mode

We also consider using different control modes for the low-level action execution. Specifically, we include both **joint-level and end-effector** actions during training and use a text identifier $c \in \{\texttt{joint}, \texttt{ee}\}$ to designate the control mode in the prompt. Then at runtime, we can pick the control mode depending on the task.

Full prompt and training details

```
<Multi-view observation><Multi-view subgoals> Task: peel vegetables. Subtask:
pick up the peeler. Speed: 8000. Quality: 5. Mistake: false. Control Mode:
joint.<Proprioception>
```

Combining all of the context information together, the example above illustrates a potential prompt that may be provided to the model.

During training, we randomly drop out each part of the prompt, which provides $\pi_{0.7}$ with the flexibility to use any subset of the prompt components at test time (e.g., running with or without subgoal images). First, we find that the model trains significantly faster when given the subgoal images — the action prediction task essentially becomes an "inverse dynamics" problem inferring the robot action between the current and future frames. Thus we only add the visual subgoal images to 25% of the examples in each batch in training. Among the examples with subgoal images we also drop out the subtask instruction $\hat{\ell}_t$ 30% of the time as often visual subgoal can substitute the equivalent textual subtask description in richer details. For episode metadata, we drop it entirely 15% of the time, and additionally each component (overall speed, overall quality, and mistake label) is dropped with 5% probability individually. We do not apply dropout for the control mode.

7.6 The $\pi_{0.7}$ Model and Training Recipe

We now discuss how we incorporate the different context in $\pi_{0.7}$ model by training on diverse data, as well as the details about the model architecture, training, and inference.

Training datasets

The training dataset for $\pi_{0.7}$ consists of demonstration data for a wide range of tasks with many different robot platforms (both static and mobile, with single arm or bimanual) in diverse environments (in-house lab-like and home-like environments, and in-the-wild home environments), autonomous data from a large amount of policy evaluations, human interventions within policy rollouts, open-source robot datasets, egocentric human video data, and auxiliary non-robot data sources from the web, including object localization and

attribute prediction, visual question answering, and text-only prediction. We also include video-language tasks including video captioning of in-house robot data and from the web.

In a significant departure from classic VLA training pipelines, we make heavy use of suboptimal robot data in training. This includes both lower quality demonstrations (failure episodes or success episodes with a substantial amount of mistakes) and data collected by prior versions of our models during model evaluation experiments[1]. For example, we use data collected by the $\pi^*_{0.6}$ model during RL training as additional examples, effectively allowing $\pi_{0.7}$ to distill their behavior. Incorporating the episode metadata into the context allows our model to effectively use all of this evaluation data and, as we will see in Sec. 7.9, enables it to attain similar performance as models that are specialized for high performance on individual tasks with RL. This corresponds to a kind of "distillation" process, where the generalist $\pi_{0.7}$ model can inherit the capabilities of RL-trained specialists. Suboptimal data also diversifies the possible states and scenarios in a given task and leads to stronger robustness, enabling the model to even sometimes outperform RL-trained or generally, single-task post-trained policies, in highly dexterous tasks.

Model architecture

The major architectural modifications of $\pi_{0.7}$ model compared to previous $\pi_{0.5}$ and $\pi_{0.6}$ models include the use of the history vision encoder from `MEM` [322] and visual subgoal images in the context. The model takes as input up to four camera images (front view, two wrist views, and optionally rear view), each with up to six history frames, and up to three subgoal images (omitting the rear view). The history frames are processed through the vision encoder and compressed to the same number of tokens as a single frame; subgoal images are processed through the same encoder. Both the camera observations and subgoal images are first resized to 448x448 pixels. For sampling history frames we use a stride of 1 second, and the entire history frames are dropped out entirely with probability 0.3. The rear view image (when available) is dropped out with probability 0.3 as well.

We employ a block-causal masking scheme, such that the observation tokens and the subgoal image tokens use bidirectional attention within themselves, and goal-image tokens can additionally attend the observations. The following text tokens use causal attention (see attention mask visualization in the appendix). We also feed the proprioceptive state $\mathbf{q}_t$ (including the history states) of the robot into the model backbone. Unlike $\pi_{0.6}$ that uses discretized text tokens to represent $\mathbf{q}_t$, $\pi_{0.7}$ follows `MEM` and embeds the state using a linear projection that maps the state dimension to the backbone dimension. Each history state is treated as an individual token; if the history frame is dropped out, the corresponding state token is masked out as well.

[1]We exclude autonomous data collected in any generalization-focused evaluation task (including ones in Sec. 7.9) from training.

The more lightweight "action expert" is a 860M-parameter transformer that is trained to predict continuous actions using flow matching objective. We use adaptive RMSNorm to inject timestep information for flow matching. The number of action tokens processed by the action expert is fixed at 50, representing an action chunk of 50 steps. The 50 tokens attend bidirectionally to each other and can also attend to the VLM backbone activations.

$\pi_{0.7}$ also employs the training-time version of real-time action chunking (RTC) [29, 34] for generating smooth action trajectories in the presence of inference delay. During training, we simulate delays of 0 to 12 timesteps, corresponding to a maximum inference latency of 240ms on a 50Hz robot.

Training with subgoal images

When training $\pi_{0.7}$ to handle subgoal images, we need the model to accommodate goals with different delays and different levels of image quality, including images generated by our world model. This requires carefully selecting which subgoals are provided as context to the model when training. We train on a combination of real images from future timesteps of the training trajectory and generated images. We found the following sampling scheme to be effective for selecting the timesteps for the real images: with probability 0.25, we sample the end-of-segment images (consistent with the prediction target for the world model), and with probability 0.75 we sample future images uniformly from 0–4 seconds ahead of the current timestep. In addition to these real images, we mitigate the train-test mismatch between real and generated images by also sampling a large number of subgoal images from the world model, and constructing additional training examples with these *generated* images added into the context of $\pi_{0.7}$ instead of the real future images.

7.7 Prompting $\pi_{0.7}$ at runtime

At runtime, we configure $\pi_{0.7}$ to run with different forms of context depending on the desired behavior **without any task-specific post-training**. For any task we always prompt the model with the control mode and episode metadata. For choosing the episode metadata, we follow

- Overall speed: set per-task to the 15^{th} percentile of the episode length from the task.
- Overall quality: always set to 5, which is the highest score.
- Mistake: always set to false, meaning no mistake.

The subtask instruction $\hat{\ell}_t$ is provided either by a learned high-level language policy or by a human supervisor for coaching (see Sec. 7.5). When the subgoal images are used, we

Algorithm 3 Prompting $\pi_{0.7}$ at test time

Input: initial observation $\mathbf{o}_0$, task instruction ℓ, episode metadata m, control mode c
Initialize subtask $\hat{\ell}$ (from high-level policy or coaching)
$\mathbf{g}^\star \sim p_\psi(\mathbf{g}^\star \mid \mathbf{o}_0, \hat{\ell}, m)$
$\mathcal{C} = \{\ell, \hat{\ell}, \mathbf{g}^\star, m, c\}$
$\mathbf{a}_{t:t+H} \sim \pi_\theta(\mathbf{a} \mid \mathbf{o}_{t-T:t}, \mathcal{C})$ ▷ Optional: CFG
for $t = 0, 1, 2, \ldots$ **do**
    if $\hat{\ell}$ changed **or** Δ-second timer elapsed **then**
        $\mathbf{g}^\star \sim p_\psi(\mathbf{g}^\star \mid \mathbf{o}_t, \hat{\ell}, m)$ ▷ Non-blocking (async)
        $\mathcal{C} = \{\ell, \hat{\ell}, \mathbf{g}^\star, m, c\}$
    if $\hat{H}$ steps elapsed since last inference **then**
        $\mathbf{a}_{t:t+H} \sim \pi_\theta(\mathbf{a} \mid \mathbf{o}_{t-T:t}, \mathcal{C}, \mathbf{a}_{t:})$ ▷ Async w/ RTC
    Execute $\mathbf{a}_t$

refresh the subgoal images whenever the semantic intent changes (i.e., new $\hat{\ell}_t$), or after $\Delta = 4$ seconds have elapsed since the last subgoal image was produced, whichever happens first. See Algorithm 3 for the full workflow. We apply asynchronous inference: the visual subgoal and subtask instruction generation happens in separate threads and the VLA inference always uses the latest ones available.

For all experiments we use 5 denoising steps to generate the 50-step action chunks and execute $\hat{H} \in \{15, 25\}$ steps out of the chunk. Since each prompt component is trained with dropout, $\pi_{0.7}$ can also be used with classifier-free guidance (CFG) [123] for any part of the prompt, for example to guide the generated actions toward higher speeds. Concretely, each action denoising step follows

$$\nabla_{\mathbf{a}} \log \pi_\theta(\mathbf{a}_{t:t+H}|\mathbf{o}_t, \mathcal{C}_t) + \\ \beta(\nabla_{\mathbf{a}} \log \pi_\theta(\mathbf{a}_{t:t+H}|\mathbf{o}_t, \mathcal{C}_t) - \nabla_{\mathbf{a}} \log \pi_\theta(\mathbf{a}_{t:t+H}|\mathbf{o}_t, \mathcal{C}_t^{\text{uncond}})),$$

where $\mathcal{C}_t^{\text{uncond}}$ denotes the set of context used in "unconditional" mode and β is the CFG weight. While any part of the context could be dropped out, we apply CFG on the episode metadata to elicit strong performance in dexterous tasks. We use moderate values of $\beta \in \{1.3, 1.7, 2.2\}$.

7.8 Robot system details

We deploy $\pi_{0.7}$ in a variety of robot platforms (Fig. 7.4), including bimanual mobile manipulators with two 6 DoF arms, static bimanual manipulators with lightweight 6 DoF arms ("BiPi"), and a bimanual UR5e system with Robotiq grippers, which we use for cross-embodiment experiments. Additional generalization and language following experiments use a single-arm 6 DoF system, using the same arms as the BiPi platform. Note that while a large fraction of

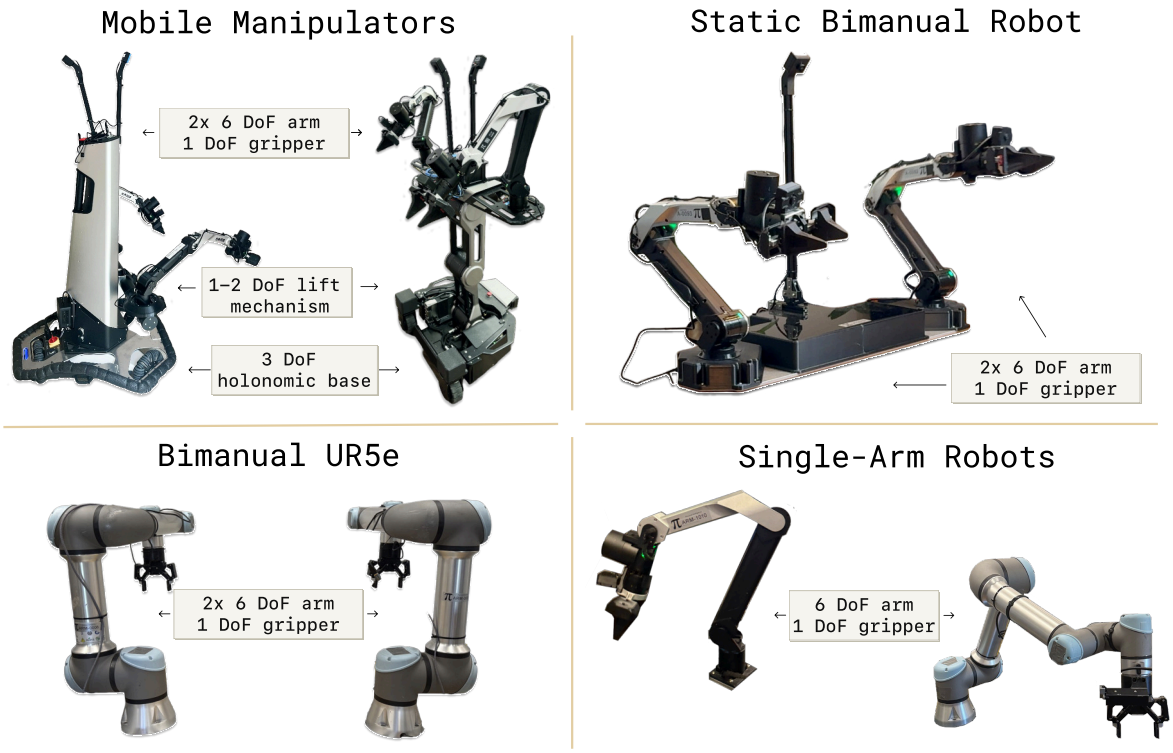


Figure 7.4: **Illustrations of some of the robots in our experiments.** We evaluate $\pi_{0.7}$ on a variety of robots, including bimanual mobile manipulators (left), static bimanual robots (middle), and a bimanual UR5e setup (right) that we use for cross-embodiment experiments.

our data is collected with arms that resemble the BiPi platform, the UR5e arms that we use for cross-embodiment testing are significantly longer, have a different morphology, and are much heavier. In practice, the UR5e arms need to employ a different manipulation strategy due to the shape of the arms, their positioning over the table (on the sides rather than at one edge), and the shape of the gripper and fingers, making cross-embodiment transfer to this platform a significant challenge. All manipulators use parallel-jaw grippers. The UR5e robots run at 20 Hz, while all other robots run at 50 Hz. Each robot has a front-facing camera as well as wrist camera on each arm, and the mobile robots also have a rear-facing camera. The action output of the $\pi_{0.7}$ model is applied on each robot using a simple PD controller. For commanding end-effector movement, we apply numerical inverse kinematics to convert target end-effector poses into target joint positions.

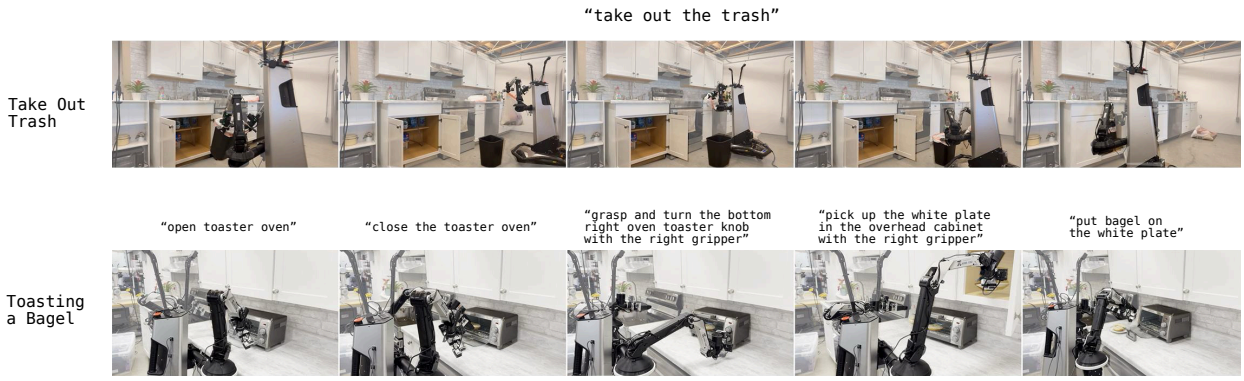


Figure 7.5: **Illustration of selected evaluation tasks.** We evaluate $\pi_{0.7}$ on a number of tasks, and two of the more longer-horizon ones are visualized here. For some tasks such as "Take Out Trash", we provide a coarse instruction like "take out the trash" and $\pi_{0.7}$ performs the full long-horizon task. For other tasks which do not appear in the training data for $\pi_{0.7}$ such as "Toasting a Bagel", we can leverage the strong language following capabilities of $\pi_{0.7}$ to coach it to perform the task with a series of detailed instructions that break down the task step-by-step.

7.9 Experimental Evaluation

In our experiments, we evaluate how well $\pi_{0.7}$ can leverage diverse data sources to enable strong out-of-the-box performance, broad generalization, and more effective transfer, leveraging a variety of context modalities. Specifically, we study how well $\pi_{0.7}$ can perform complex and dexterous tasks out of the box, particularly in comparison with more specialized RL-finetuned models (Sec. 7.9), evaluate its ability to flexibly follow instructions to do a variety of different tasks (Sec. 7.9), study its transfer capabilities across embodiments (Sec, 7.9), and test its ability to compose skills in previously unseen ways to do new tasks (Sec. 7.9). Finally, we perform controlled experiments to study how the performance of $\pi_{0.7}$ scales with increased task and context diversity in our robot datasets (Sec. 7.9).

Out-of-the-box performance on challenging tasks

$\pi_{0.7}$ achieves high performance on dexterous tasks without task-specific post-training. In our first set of experiments, we study how well $\pi_{0.7}$ can master dexterous tasks that were seen in the training data, but where the goal is to perform these tasks as robustly and efficiently as possible. This is surprisingly difficult for prior robotic foundation models: often the best-performing policies are fine-tuned for specific downstream tasks, even if they use generalist pre-training [317, 316]. We aim to answer: can the general-purpose $\pi_{0.7}$ model match the performance of task-specific fine-tuned models on a variety of dexterous manipulation tasks?

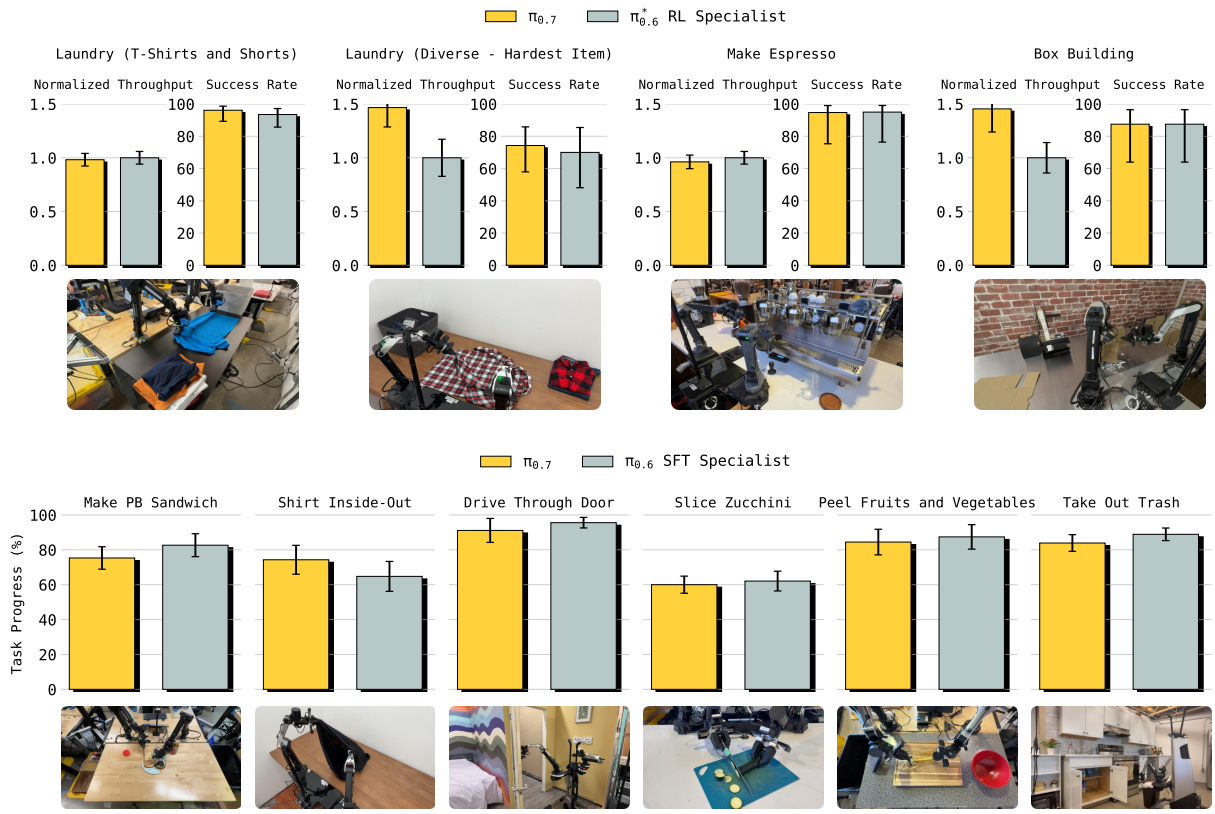


Figure 7.6: **Out-of-the-box dexterity:** $\pi_{0.7}$ can perform a wide range of highly dexterous tasks directly out of the box. We consider tasks from $\pi^*_{0.6}$ [317] (top row) and a number of other dexterous tasks including ones from the "Robot Olympics" experiments (bottom row). For the tasks from $\pi^*_{0.6}$, we report success rate and normalized throughput (relative to the specialist model; raw throughput means successes per hour), while for other tasks we report task progress. We find that the same $\pi_{0.7}$ model can match the performance of the task-specific post-trained specialist policy from $\pi^*_{0.6}$ or $\pi_{0.6}$ for each of these tasks, and even achieve higher throughput than the RL specialists in diverse laundry folding and box building.

We use the tasks shown in Fig. 7.6. These include the espresso making, box building, and laundry folding tasks that we previously used to evaluate the RL-trained $\pi^*_{0.6}$ models [317], where we can directly compare the speed and robustness of the single general-purpose $\pi_{0.7}$ model to the individual RL-finetuned specialist $\pi^*_{0.6}$ models. We also study a number of other dexterous tasks, including some tasks from our previous "Robot Olympics" experiments (making a peanut butter sandwich, turning a shirt inside-out, and driving through a door) and a number of additional dexterous tasks, such as fully slicing up a zucchini, peeling a few fruits and vegetables (zucchini, cucumbers, and carrots), and a long-horizon task that

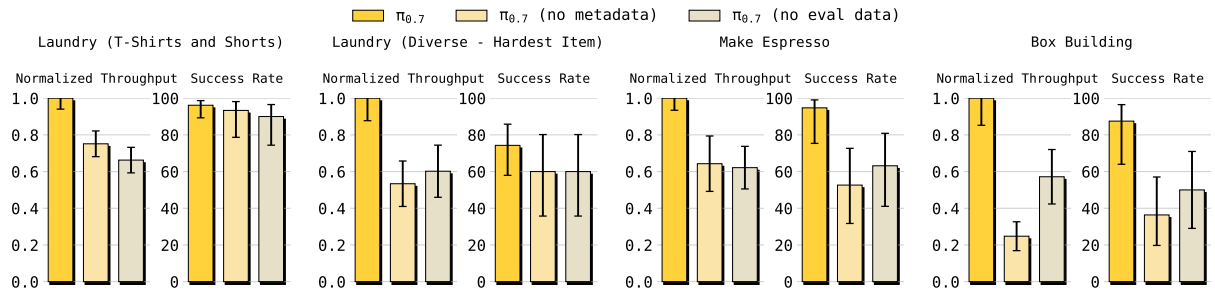


Figure 7.7: **Impact of prompt composition and evaluation data on out-of-the-box performance:** We compare $\pi_{0.7}$ with two ablations: one that does not include episode metadata in the context, $\pi_{0.7}$ (no metadata), and another that does not include data from autonomous evaluation episodes during training, $\pi_{0.7}$ (no eval data). We find that $\pi_{0.7}$ outperforms both $\pi_{0.7}$ (no metadata) and $\pi_{0.7}$ (no eval data) across the board, with the gap most prominent in throughput. Throughput (successes/hour) here is normalized relative to $\pi_{0.7}$.

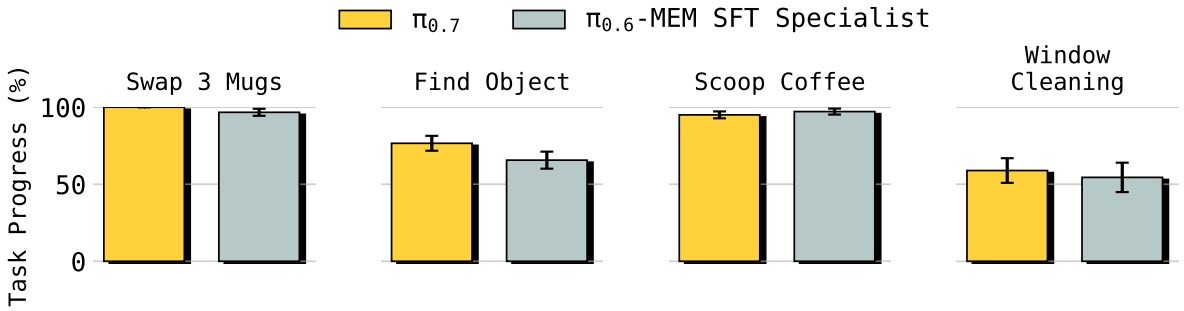


Figure 7.8: **Tasks that require memory:** $\pi_{0.7}$ can also perform tasks that require explicitly keeping track of prior context, achieving similar or better performance compared to the specialist policies with memory fine-tuned to some of the tasks in the MEM paper [322].

involves replacing a trash bag in a trash can. We find that $\pi_{0.7}$ achieves performance that is competitive with the RL specialists used in the $\pi^*_{0.6}$ release [317] for all of the tasks considered in the paper directly out of the box (Fig. 7.6, first row), and even outperform the specialists in throughput in the difficult laundry and box building tasks. Additionally, we compare $\pi_{0.7}$ to SFT specialists trained on top of $\pi_{0.6}$ for a number of other dexterous tasks, and find that $\pi_{0.7}$ is again able to closely match the performance of all specialist policies (Fig. 7.6, second row).

To understand how the training recipe of $\pi_{0.7}$ affects performance, we additionally compare

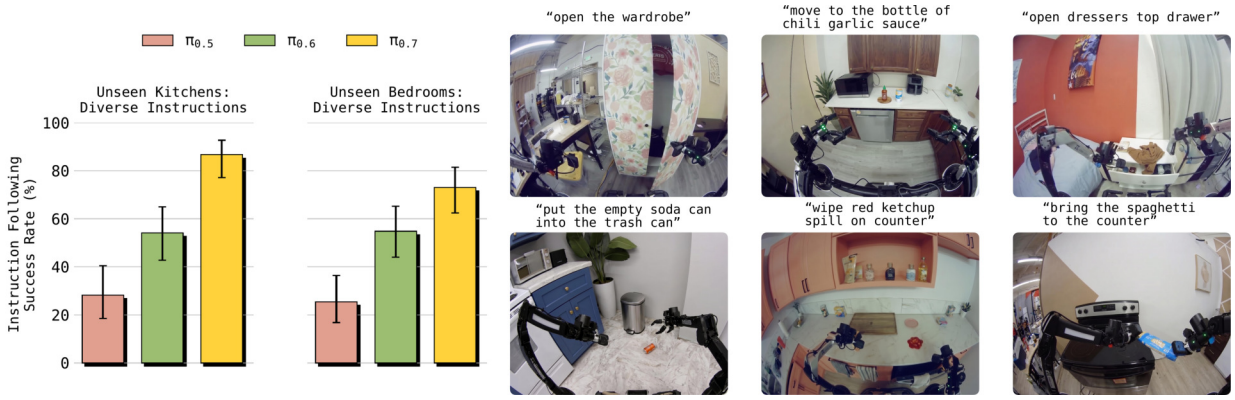


Figure 7.9: **Broad instruction following in novel environments:** We evaluate $\pi_{0.7}$ on 14 instruction following scenarios, each of which involve following a sequence of 3-6 open-ended instructions, across 4 unseen kitchen and 2 unseen bedroom environments. We report the instruction following success rate, the percentage of total instructions that were correctly followed across all evaluations. We find that $\pi_{0.7}$ significantly outperforms $\pi_{0.5}$ and $\pi_{0.6}$ across the board, attaining high absolute success rates.

$\pi_{0.7}$ with two ablations on the tasks from the $\pi^*_{0.6}$ release: $\pi_{0.7}$ (no eval data), which holds out all autonomous evaluation episodes from training (and thus cannot benefit from distilling agent rollouts from strong, e.g. RL trained, policies), and $\pi_{0.7}$ (no metadata), which omits episode metadata from the context as shown in Fig. 7.7 for the tasks used in the $\pi^*_{0.6}$ release. Results suggest that $\pi_{0.7}$ significantly outperforms both $\pi_{0.7}$ (no eval data) and $\pi_{0.7}$ (no metadata) on all tasks. Since policy evaluation data can vary widely in quality, training on this data, combined with rich metadata to disambiguate high and low quality behaviors, is critical for $\pi_{0.7}$'s strong performance on all of these challenging tasks.

$\pi_{0.7}$ achieves high performance on tasks that require memory without fine-tuning. In these experiments, we study how well $\pi_{0.7}$ can perform tasks that require explicitly keeping track of previous observations [322]. We compare the same single $\pi_{0.7}$ model out of the box to the task-specific fine-tuned versions of $\pi_{0.6}$ with memory used in Torne et al. [322], and find that $\pi_{0.7}$ can achieve similar or better performance to the fine-tuned specialists on all of these tasks (Fig. 7.8).

Instruction following

In the next set of experiments, we study how well $\pi_{0.7}$ can follow language instructions, including its ability to carry out a variety of different contexts and follow referential instructions that differ systematically from the training data. Our experiments focus specifically on performing tasks in messy environments where there are many possible tasks that the robot could perform, requiring $\pi_{0.7}$ to pay careful attention to the provided instruction in order

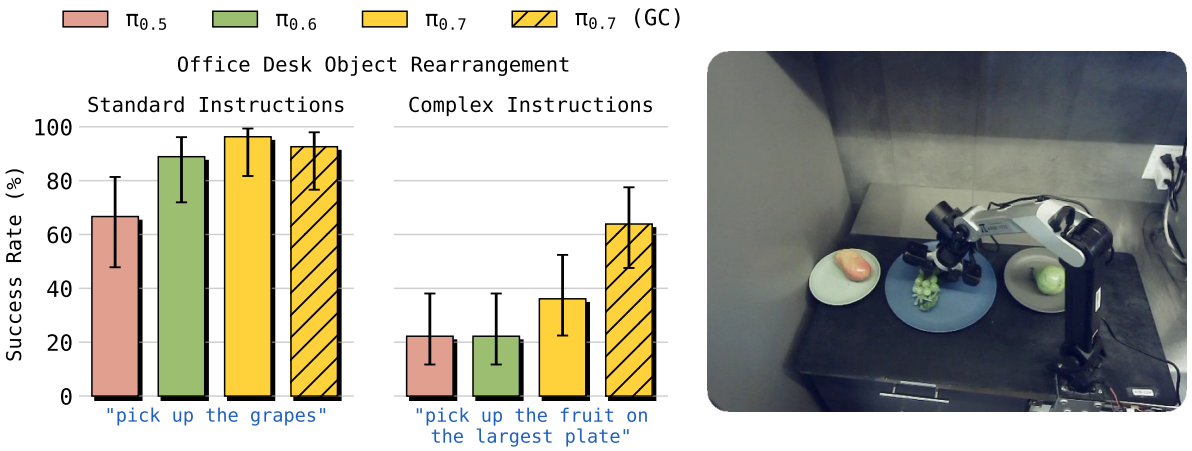


Figure 7.10: **Following complex referential instructions:** $\pi_{0.7}$ and prior models all succeed on the simpler re-arrangement instructions (instructions include "pick up the spoon", "put the spoon to the left of the fork" and "put the spoon to the right of the fork"), but $\pi_{0.7}$ performs significantly better on the complex and unusual instructions (instructions include "pick up the largest bowl on the table", "pick up the object I would use to eat soup" and "pick up the fruit on the largest plate"). Including subgoal images generated by a lightweight world model ($\pi_{0.7}$ (GC)) further boosts instruction following performance, making $\pi_{0.7}$ significantly more capable at following complex instructions.

to succeed. We find that $\pi_{0.7}$ displays instruction following capabilities that significantly improve upon our prior models $\pi_{0.5}$ [31] and $\pi_{0.6}$ [316].

$\pi_{0.7}$ can be flexibly prompted to perform a wide variety of different tasks. Language following has presented a notorious challenge for robotic foundation models, particularly with open-vocabulary instructions that do not correspond directly to ones seen in training. In these experiments, we aim to study the breadth of $\pi_{0.7}$'s capabilities, to answer: can $\pi_{0.7}$ better handle a wider variety of language instructions than prior models?

We evaluate $\pi_{0.7}$ on a diverse set of instructions across 4 unseen kitchens and 2 unseen bedrooms that were not present in the training data (Fig. 7.9). Each experiment tests whether the robot can follow a 3 to 6 step sequence of instructions to achieve a specific goal. The tasks require a variety of different realistic tasks in kitchen and bedroom environments, including re-arranging and tidying items, interacting with furniture, and cleaning up spills. This combination of new test environments and diverse instructions presents a major challenge for robotic foundation models, which can struggle to follow even simple instructions in seen environments. We find that $\pi_{0.7}$ is able to significantly outperform $\pi_{0.5}$ and $\pi_{0.6}$, with a high

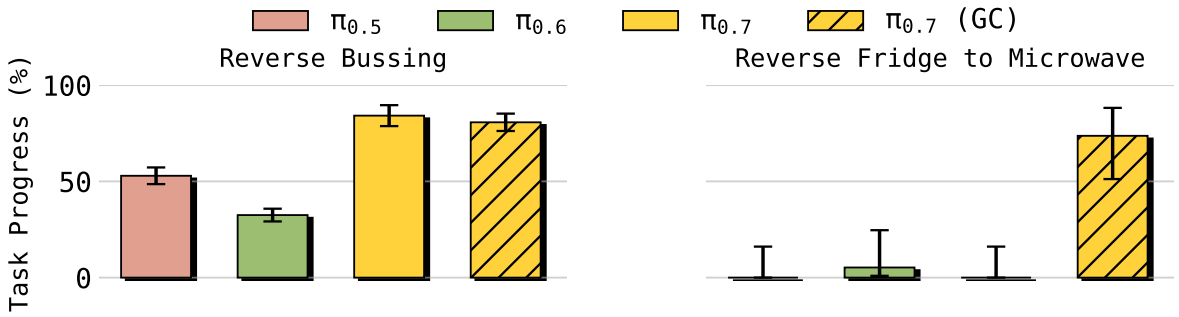


Figure 7.11: **Breaking dataset biases by following instructions:** the improved language-following performance of $\pi_{0.7}$ enables it to break strong dataset biases. Prior models struggle on these data bias challenge tasks, which require following instructions that contradict the pattern in the data (e.g., putting dishes in the trash and trash in the dish bin). $\pi_{0.7}$, however, can follow instructions sufficiently well that it can break these strong biases to still perform the task. Notably, for the "Reverse Fridge to Microwave" task, including subgoal images from the world model in the context ($\pi_{0.7}$(GC)) is critical for success.

overall instruction following success rate.

$\pi_{0.7}$ can handle out-of-distribution referential instructions. Because of the breadth and diversity of our training data, it is generally difficult for us to quantify how *novel* the test instructions are. In the next set of experiments, we intentionally designed a set of instructions that are unusual, refer to objects in unconventional ways, or require understanding spatial relations. In Fig. 7.10, we compare $\pi_{0.7}$ and prior models on a set of object re-arrangement instructions, broken up into *standard* and *complex* instructions. The standard instructions are phrased in a similar way to language instructions in the training data. The complex instructions use unusual language or complex spatial references, such as "pick up an object I would use to eat soup" or "pick up the fruit on the largest plate." In these experiments, we also evaluate $\pi_{0.7}$ with and without subgoal images, which are generated by a lightweight world model as discussed previously. We can see that $\pi_{0.7}$ improves over prior models on the complex instructions, and the use of subgoal images further boosts its performance, importing semantic understanding from the world model.

$\pi_{0.7}$ can follow instructions that go against dataset biases. Dataset bias presents a major challenge for instruction following: if the robot always does the same thing in a given scene, a model trained on such data will often ignore language in these scenes, blindly copying the behavior seen in the data. In the next experiment, we construct scenarios that suffer from this problem, and test if we can prompt $\pi_{0.7}$ to go *against* the natural bias in the dataset. We constructed two tasks: "Reverse Bussing" and "Reverse Fridge to Microwave." In

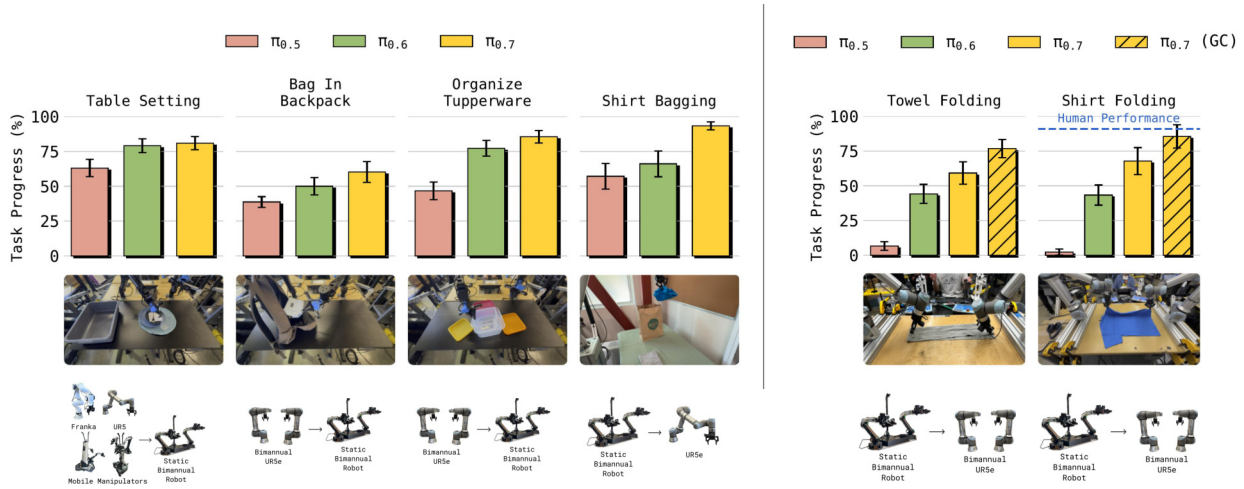


Figure 7.12: **Cross-embodiment transfer:** Left: Both $\pi_{0.7}$ and prior models achieve strong cross-embodiment transfer directly out of the box on simpler re-arrangement or repositioning style tasks. For example, for the "Table Setting" task, data was collected with a variety of different robots, and the task was tested on the static bimanual robot. All of the models performed well. For tasks that required transfer from the bimanual UR5e robot to the smaller static bimanual robot ("Bag In Backpack" and "Organize Tupperware"), the embodiment gap is larger, as there is only one source robot that is bigger and heavier than the target robot. Here, $\pi_{0.5}$ performed very poorly, but $\pi_{0.6}$ still performs quite well. Transferring from the smaller static bimanual robot to the UR5e for the "Shirt Bagging" task introduces the largest embodiment gap, and here the $\pi_{0.7}$ model significantly outperforms prior models. Right: for the more dexterous tasks that require folding towels and t-shirts, the embodiment gap poses an even greater challenge. Data for these tasks was collected with the smaller static bimanual robot, and the task was evaluated on the larger bimanual UR5e platform. $\pi_{0.7}$ was able to transfer this task successfully, and improved even more when using visual subgoal images generated by our lightweight world model. In fact, task progress matches the "zero-shot" performance of our most experienced human teleoperators, who have operated a diverse set of robots and attempted this task on the UR5e for the first time.

our dataset, the "bussing" task involves putting trash in a trash bin and dishes in a bussing bin. For the "Reverse Bussing" task, the robot is asked to do the opposite: put the trash in the bussing bin, and the dishes in the trash. The "Fridge to Microwave" task requires taking food out of the fridge and putting it in the microwave, and we did not collect data going the other way. At test time, in the "reverse" version of this task ("Reverse Fridge to Microwave"), we prompt the robot to take the food from the microwave to the fridge, violating the bias in the dataset.

The results in Fig. 7.11 show that $\pi_{0.7}$ significantly improves over prior models on these

tasks. This suggests that $\pi_{0.7}$ has significantly better language following capabilities, and pays enough attention to the instructions to overcome the bias in the data for these tasks. On the "Reverse Fridge to Microwave" task, conditioning on generated subgoal images ($\pi_{0.7}$ (GC)) is critical for success, as the world model can generate subgoals based on textual instructions effectively by leveraging web-scale image generation pre-training.

Cross-embodiment transfer

While a number of models have used data from multiple different robot embodiments [240, 350, 315, 331], zero-shot transfer of complex tasks from a source embodiment to a target robot that has never seen the task presents a major challenge. In these experiments, we aim to study whether cross-embodiment transfer is an *emergent* property of $\pi_{0.7}$. Namely, can $\pi_{0.7}$ directly transfer capabilities to robot embodiments where no task-specific data was ever collected?

We find that for several tasks, $\pi_{0.7}$ succeeds on target embodiments entirely out of the box, on tasks for which the target robot has no training data. For more modest embodiment differences, the $\pi_{0.5}$ and $\pi_{0.6}$ models also show some amount of emergent cross-embodiment transfer. However, as the gap in the robot morphology increases, effectively transferring complex skills requires more significant changes in strategy. In these cases, $\pi_{0.7}$ significantly outperforms the prior models, and even matches the "zero-shot" performance of human teleoperators for the shirt folding task, as we discuss below. Experimental results below apply joint-space control, as we find that end-effector control does not yield noticeable gains in performance with our prior models (Appendix F.5).

Zero-shot cross-embodiment transfer for object re-arrangement tasks. We first study a set of simpler object rearrangement tasks, where we test $\pi_{0.7}$ on a robot other than the one that was used to collect data for that task. The results are presented in Fig. 7.12. For the first task, "Table Setting," the data was collected with several different robot types, including mobile, static, and single-arm systems. This is the most favorable setting for cross-embodiment transfer, since the models can infer the common structure of the task from multiple different robots. When evaluated on a static bimanual robot system, we find that all methods show strong signs of cross-embodiment transfer. However, when we increase the embodiment gap more significantly and test policies on the smaller static bimanual platform when all data was collected on the larger UR5e bimanual platform ("Bag In Backpack" and "Organize Tupperware" tasks), we find that the performance of $\pi_{0.5}$ degrades significantly, while both $\pi_{0.6}$ and $\pi_{0.7}$ still are able to achieve strong performance. We then increase the embodiment gap even more, studying transfer for a task where data was collected on the smaller static bimanual platform and evaluated on a *single-arm* UR5e platform ("Shirt Bagging" task). This necessitates a significantly different strategy, as the target robot has only one arm, is much larger, and also heavier. Here, $\pi_{0.7}$ significantly outperforms the prior models.

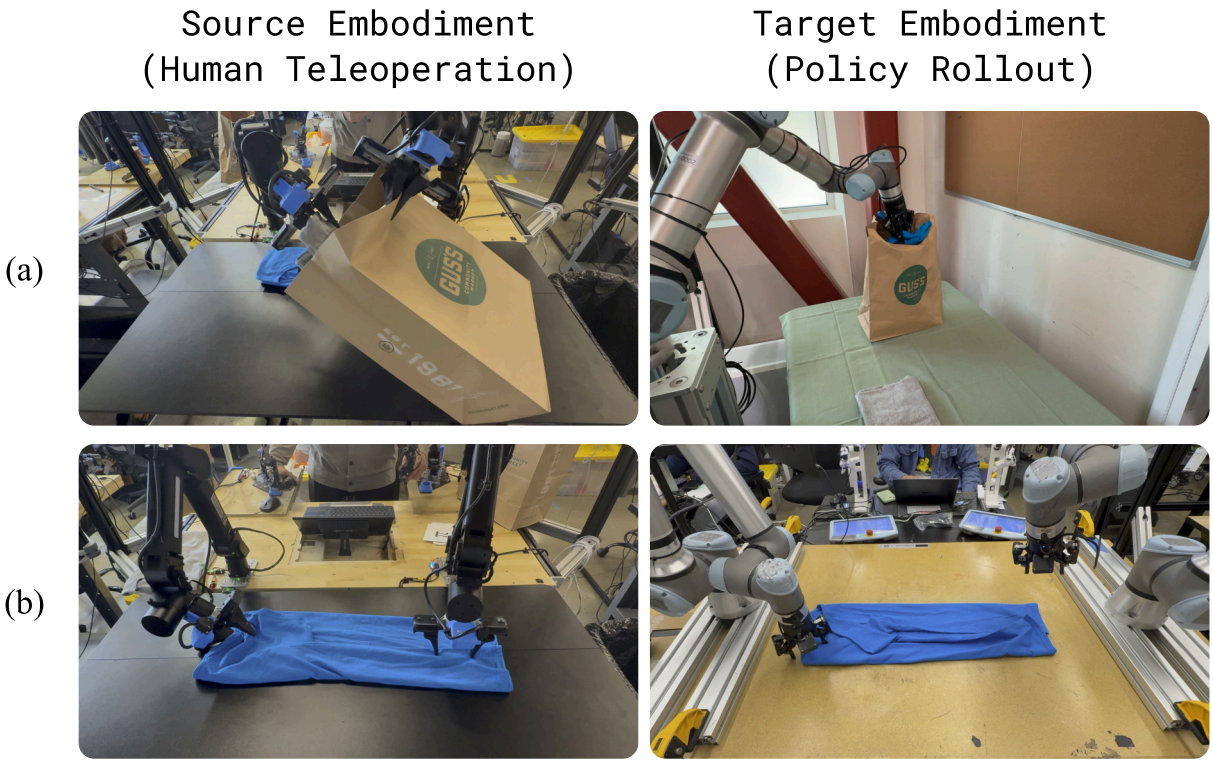


Figure 7.13: **Cross-embodiment transfer produces emergent strategies adapted to the target embodiment.** (a) On the source robot, human teleoperators use one arm to hold the bag open while the other performs insertion. On the UR5e target robot, $\pi_{0.7}$ instead discovers a single-arm pick-and-place strategy suited to the robot's greater reach. (b) Human teleoperators approach the shirt with a tilted end-effector on the source robot, while $\pi_{0.7}$ produces vertical grasps on the UR5e, which is more suitable for the larger robot's arm placement. In both cases, the policy goes beyond replicating source behavior, discovering manipulation strategies for the task that are better suited to the target embodiment.

Notably, successful transfer often requires the policy to discover new manipulation strategies suited to the target morphology rather than simply replicate the source behavior. For example, the shorter static bimanual robot must use one arm to hold the bag open while the other performs insertion, whereas the taller UR5e arm can accomplish the same task with a single-arm pick-and-place (Fig. 7.13 (a)). Despite this significant morphological gap, the model applies the appropriate strategy for each embodiment, demonstrating cross-embodiment transfer that goes beyond mimicking the original robot's motion.

Zero-shot cross-embodiment transfer for dexterous tasks. Dexterous tasks, such as laundry folding, present a bigger challenge for cross-embodiment transfer. Such tasks require more precise manipulation skills than just grasping and repositioning objects. Successfully



Figure 7.14: **Example of language coaching:** We can "teach" a new task to $\pi_{0.7}$ by providing step-by-step verbal instructions. Because of its language following ability, $\pi_{0.7}$ can perform new tasks successfully under user instruction, and these instructions can then by used to train a high-level policy that prompts $\pi_{0.7}$ so that it can perform the task fully autonomously.

folding a t-shirt requires a sequence of precise grasps and placements, and the correct grasp angle might vary with the robot's reachable workspace and gripper orientation. Most of our folding data was collected with lightweight static bimanual robots (see Fig. 7.4). We did not collect any laundry folding data with the bimanual UR5e system. The morphology of this system, its reachable workspace, and its dynamics (e.g., higher internal inertia) differ significantly from the robots with which we collected laundry folding data. We found the UR5e generally harder to teleoperate and less suitable for very precise grasps, suggesting that folding laundry with this robot requires a change in the manipulation strategy.

$\pi_{0.7}$ was able to successfully fold both towels and shirts on the bimanual UR5e system (Fig. 7.12, right). On the source robot (the static bimanual robot in Fig. 7.4), human operators often approach the cloth with a tilted end-effector to secure the fabric against the table before lifting. On the UR5e, $\pi_{0.7}$ instead uses a vertical grasp that is better suited to the arm's kinematics — a strategy that is different from the training data on the source robot but more suitable for the target embodiment (Fig. 7.13 (b)). We also find that using subgoal image generation with our world model significantly improves performance (labeled $\pi_{0.7}$ (GC) in the plots), as the world model can more effectively construct visual analogies between the source and target robots. The generated subgoals predict what kind of grasps and clothing configurations would be reasonable for the target robot, and the model incorporates this additional context as a hint for selecting better actions.

To contextualize these results, we conducted a human subject study with 10 experienced teleoperators (mean 375 hours of teleoperation experience across all robots, all within the top 2% by experience). Like the policy, these operators had extensive experience on the source embodiment, but had never attempted shirt folding on the UR5e bimanual system, making this a zero-shot cross-embodiment transfer setting for both humans and the policy. The human operators achieved 90.9% task progress and an 80.6% success rate, while $\pi_{0.7}$ achieved 85.6% task progress and an 80% success rate, demonstrating performance comparable to these expert operators. The strong cross-embodiment transfer performance highlighted by

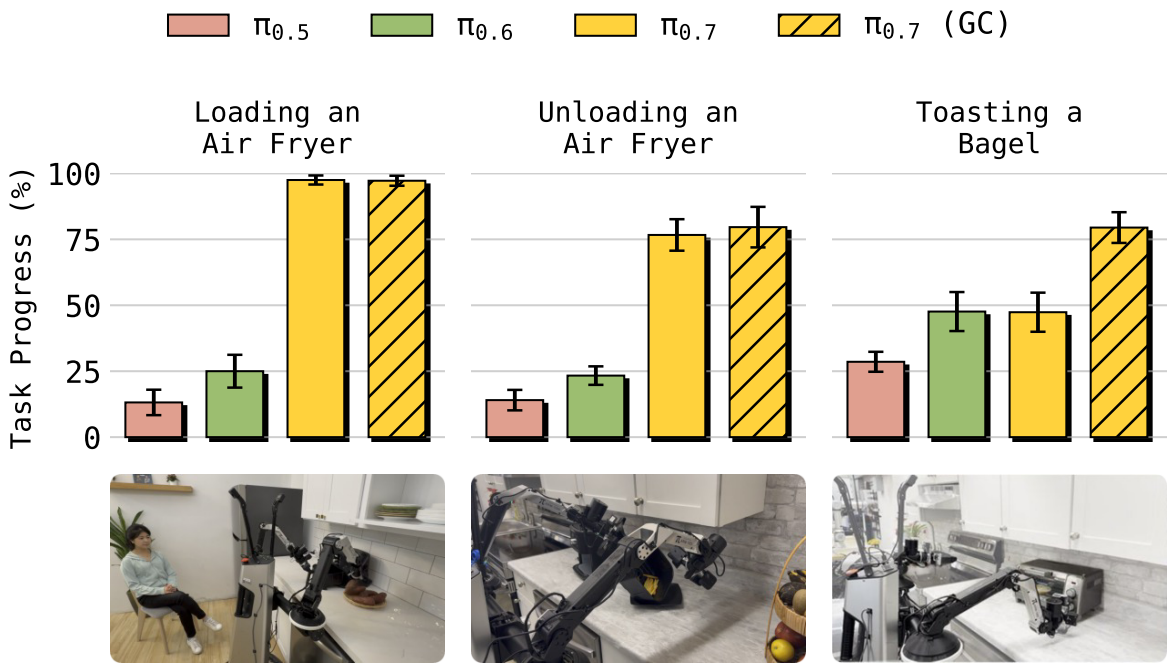


Figure 7.15: **Coaching to perform new long-horizon tasks:** Because $\pi_{0.7}$ can follow language instructions effectively, even for unfamiliar skills, it can be "coached" to perform a number of unseen, longer horizon tasks both when conditioned on language and generated subgoal images ($\pi_{0.7}$ (GC)). Prior models generally lack the language following ability needed to follow the coaching commands, and thus achieve very poor performance.

this comparison is scientifically exciting, and it has a practical implication: dexterous skills may be transferred *from* lightweight, low-cost platforms that are easy to teleoperate *to* high-payload industrial arms, where collecting human demonstration data is substantially more expensive and difficult. More details on the human study setup and results are provided in Appendix F.6.

Compositional task generalization

In the next set of experiments, we study how well $\pi_{0.7}$ can perform new tasks by compositionally generalizing over the skills seen in training. This has been viewed as a kind of "grand challenge" for robotic foundation models: while prior models have demonstrated generalization over semantic concepts (e.g., reaching for an object with an unseen textual label), performing new tasks has proven elusive.

We found that for some short-horizon tasks, $\pi_{0.7}$ can work well completely out of the box,

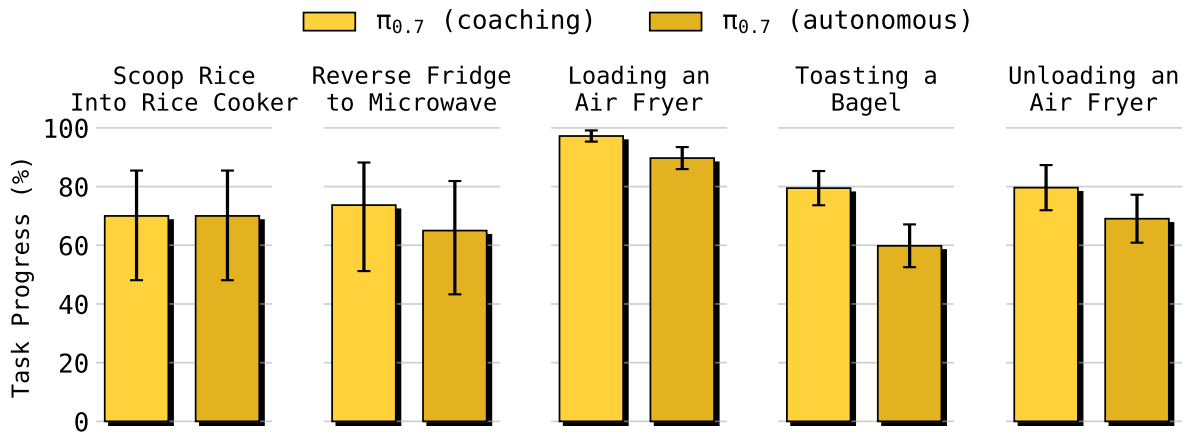


Figure 7.16: **Acquiring new autonomous capabilities with coaching:** We can use the coaching episodes collected for a number of different unseen tasks to train a high-level policy to automatically prompt $\pi_{0.7}$ in accordance with the coaching episodes. This allows us to create fully autonomous policies for these tasks ($\pi_{0.7}$ (autonomous)) that closely match the performance of the policy with live human coaching ($\pi_{0.7}$ (coaching)) *without collecting any additional data with teleoperation or any other kind of low-level actions.*

despite no data being collected explicitly for these tasks. These tasks involve manipulating unfamiliar objects in new ways, such as wiping down a pair of headphones with a cloth, or spinning a desk fan (Fig. 7.17). For longer-horizon, more complex tasks, we find that we can actually use the instruction following capabilities of $\pi_{0.7}$ to "coach" it through the task with language. This provides an exciting new way to teach $\pi_{0.7}$ new tasks without collecting any additional training data.

$\pi_{0.7}$ can perform new short-horizon tasks out of the box. We find that $\pi_{0.7}$ can perform short-horizon tasks such as pressing the plunger on a french press, scooping rice into a rice cooker, wiping down common office objects, and spinning various articulated items directly out of the box, despite no robot data being collected specifically for any of these tasks (Fig. 7.17). This is quite exciting — since $\pi_{0.7}$ can flexibly compose a wide variety of skills, we can often get it to perform new, simple tasks by just prompting it to get the desired behaviors.

$\pi_{0.7}$ can be coached to perform new longer horizon tasks purely with language. While $\pi_{0.7}$ can be directly prompted to perform new short-horizon tasks, simply asking the robot to perform an unseen long-horizon task like cooking a sweet potato would not work: although $\pi_{0.7}$ exhibits a high level of out-of-the-box generalization, these tasks are simply too complex, requiring up to 5 minutes of interaction with multiple stages. However, $\pi_{0.7}$'s

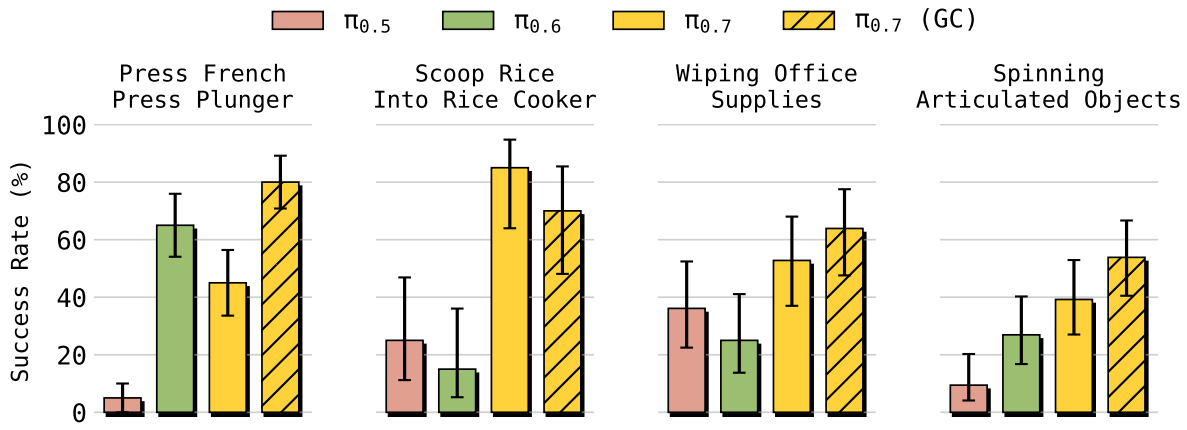


Figure 7.17: **Performing new short-horizon tasks:** $\pi_{0.7}$ can perform a number of new short-horizon tasks directly out of the box, including scooping rice into a rice cooker, spinning various objects such as a gear set and desk fan, and wiping down objects with a cloth, such as a ruler and headphones, despite no data being collected for any of these tasks. $\pi_{0.7}$ shows roughly equally strong performance when conditioned directly on language instructions ($\pi_{0.7}$) or generated image goals ($\pi_{0.7}$ (GC)).

language following abilities provide us with an exciting new path to teach the model such tasks: instead of providing demonstration data for each complex skill that we want the robot to learn, we can instead "coach" the robot to perform the new task with language, a bit like how one might teach the task to a person (Fig. 7.14). We set up several realistic multi-stage kitchen tasks: (1) "Loading an Air Fryer": using an air fryer to cook a sweet potato; (2) "Unloading an Air Fryer": dumping items out of an air fryer, and (3) "Toasting a Bagel": using a toaster to toast up a bagel. In each case, our robot data did not contain any training episodes for this task, although similar appliances were seen in different contexts in human data and external datasets. However, a person can walk the robot through task with step-by-step instructions, such as "pick up the sweet potato" and "open the air fryer." We present the results of coaching $\pi_{0.7}$ and comparisons in Fig. 7.15. Critically, none of the models have any *action-level* data of these particular tasks, and in the "coaching" episodes, the environment and task are entirely unseen. We find that $\pi_{0.7}$ can be coached much more effectively than prior methods to perform all of these tasks, and can perform even more effectively when conditioned on generated subgoal images.

Coaching data can endow $\pi_{0.7}$ with new capabilities. Since $\pi_{0.7}$ can be coached to perform new tasks, we can actually use the step-by-step instructions in the coaching data to train a high-level language policy which can prompt $\pi_{0.7}$ with the appropriate language instructions as it performs the task. This gives $\pi_{0.7}$ the ability to perform fully unseen,

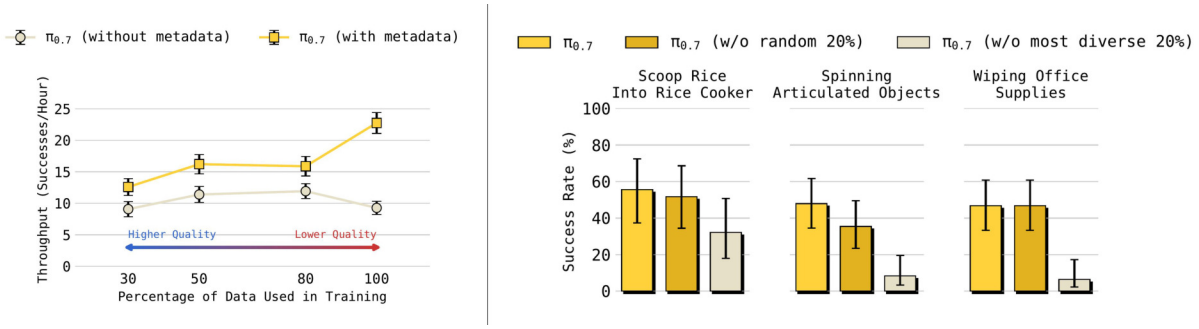


Figure 7.18: **Scaling of generalization performance with diverse context and data:** Left: We find that $\pi_{0.7}$ (with metadata) can continuously improve its performance when it is trained on larger datasets, even when the average quality of the data actually decreases. By contrast, without training on rich conditioning information, we find that $\pi_{0.7}$ (without metadata) actually can degrade in performance as more lower quality data is introduced. Right: When $\pi_{0.7}$ is trained without our robot data with the highest task diversity, its performance degrades substantially. This suggests that $\pi_{0.7}$ is able to utilize the task diversity in our robot data to drive substantial improvements in compositional task generalization.

long-horizon tasks without collecting any additional teleoperation data. We show the results of this experiment in Fig. 7.16 for five different tasks. For all of these tasks, we find that we can successfully train an autonomous policy ($\pi_{0.7}$ (autonomous)) that can roughly match the performance of the coaching episodes ($\pi_{0.7}$ (coaching)) that were collected by prompting the model to perform the task.

Can $\pi_{0.7}$ learn effectively from diverse and mixed-quality data?

In our last set of experiments, we perform a set of controlled ablation studies to understand whether $\pi_{0.7}$ can effectively leverage large and diverse datasets, and whether its performance improves with dataset diversity. These questions are difficult to answer definitively, since the performance of such a model depends on a large number of factors, and very large datasets are very difficult to slice cleanly so as to ablate "diversity". To provide some understanding of these questions, we first study whether $\pi_{0.7}$ continues to improve on seen tasks when trained on increasingly larger but more mixed-quality datasets. Then, we study whether $\pi_{0.7}$ can leverage datasets with high task diversity to drive improvements in generalization.

$\pi_{0.7}$ can effectively learn from mixed-quality data: Effectively learning from diverse robot data has so far been a significant challenge in training robotic policies. Designers will often carefully filter or curate the data to pull high-quality datasets with consistent strategies [118, 181]. However, data filtering is laborious, task-specific, and ends up throwing away a lot of valuable information. In these experiments, we aim to answer: can $\pi_{0.7}$ learn more from data with diverse manipulation strategies?

To study this question, we consider the "Laundry (T-Shirts and Shorts)" task that we studied in Fig. 7.6. We annotated the data collected by human operators based on fold quality and speed, and split our dataset into 4 buckets consisting of (1) the top 30% by quality and speed, (2) the top 50%, (3) the top 80%, and (4) all of the data. We then train new $\pi_{0.7}$ models from scratch for data from each of the four buckets, using episode metadata or without it (8 models in total). We find that while $\pi_{0.7}$ (without metadata) can actually get worse when trained on larger, mixed-quality datasets, $\pi_{0.7}$ (with metadata) is able to continuously improve as we train on more data, even though the dataset size increases corresponds to a decrease in average data quality (Fig. 7.18, left). This suggests that our diverse prompting method effectively makes the model design *more scalable*, in the sense that it benefits more from larger datasets, even when these larger datasets actually consist of lower quality data that harm models trained in the usual way. Episode metadata effectively disambiguates the different data quality and strategies within the large-scale datasets during $\pi_{0.7}$ training, and enables prompting for the desired mode of behavior at test time.

Can $\pi_{0.7}$ translate increased dataset diversity into better generalization performance?

To study this question, we compare $\pi_{0.7}$ with the following ablations on some of the short-horizon unseen tasks from Fig. 7.17.

- $\pi_{0.7}$ (w/o most diverse 20%): $\pi_{0.7}$ but with the 20% of our data with the highest task diversity removed.
- $\pi_{0.7}$ (w/o random 20%): $\pi_{0.7}$ with a randomly sampled 20% of our data removed to serve as a data-controlled comparison to $\pi_{0.7}$ (w/o most diverse 20%)

The comparison between $\pi_{0.7}$ (w/o most diverse 20%) and $\pi_{0.7}$ (w/o random 20%) allows us to understand the impact of data with high task diversity in a controlled way, as both models are trained on the same quantity of data. We find that across all tasks, $\pi_{0.7}$ and $\pi_{0.7}$ (w/o random 20%) significantly outperform $\pi_{0.7}$ (w/o most diverse 20%), demonstrating that $\pi_{0.7}$ is able to effectively ingest data with high task diversity and translate that data into performance improvements on short-horizon, unseen tasks (Fig. 7.18, right).

7.10 Discussion

We presented $\pi_{0.7}$, a general-purpose robot foundation model that exhibits out-of-the-box compositional generalization, effective language following, and task performance that is competitive with more specialized models that are fine-tuned to individual dexterous tasks. At the core of $\pi_{0.7}$ is a diverse prompting strategy, inspired by prompt expansion, where

additional information about the episode is provided to the model during training and, optionally, at test time. This additional information includes more detailed language, episode metadata, and subgoal images. Our experiments show that by using diverse prompting and a larger and more diverse dataset, $\pi_{0.7}$ can represent policies of many different qualities, and distill specialist performance back into one pre-trained model. $\pi_{0.7}$ acquires a number of emergent capabilities, such as the ability to transfer skills across robots, follow complex language commands, and generalize compositionally, recombining skills in new ways to solve new tasks.

While $\pi_{0.7}$ generalizes broadly, the success rate for zero-shot generalization is (unsurprisingly) lower than in-distribution tasks: while seen tasks often have success rates in excess of 90%, unseen tasks or unseen task-robot combinations have success rates in the 60-80% range. An exciting direction for future work is to leverage the high steerability of $\pi_{0.7}$ to efficiently learn from data in the test task, for example with more detailed language coaching or even with autonomous reinforcement learning.

A perhaps surprising limitation of our experiments is that it's practically difficult when training on such large and diverse datasets to definitively determine which tasks are truly "seen" or "unseen": while some of our generalization experiments (e.g., those in Sec. 7.9) use tasks for which we did not deliberately collect data, our dataset contains so many different scenes and behaviors that potentially related skills may well be present elsewhere in the data, either with a different label, or incidentally as part of performing other tasks. In many ways this mirrors the challenge of understanding generalization with large language models: determining what is truly novel becomes difficult, and the model may well be achieving generalization primarily by "remixing" skills and behaviors from other situations. However, we would posit that this in fact is the essence of *compositional generalization*. Practically, whether the behaviors are truly new or merely novel combinations of seen parts, the ramifications are similar: instead of deliberating collecting targeted data for each new task that we want the robot to solve, a model that provides compositional generalization would allow the user to simply *prompt* it to do the desired task. Models that can enable such compositional generalization at scale would transform how we approach robotic learning, making it possible to prompt, coach, and explain things to a robot rather than needing to collect additional action data.

Chapter 8

Conclusion

In this dissertation, we presented approaches for training and deploying generalizable, scalable robot policies. In Part I, we demonstrated the efficacy of robot pre-training. Early language models — such as BERT and GPT-2 [73, 262] — were most effective not as out-of-the-box reasoners, but as initializations for solving downstream tasks via fine-tuning. It stands to reason that early generalist robot policies would be the same way: not strong enough to solve many tasks zero-shot, but strong enough to outperform *non-robot* pre-training (e.g., visual representation alignment) in a fine-tuning setting. Indeed, we were able to validate this hypothesis, first at a smaller scale with an encoder-decoder model in Chapter 2, and then at a larger scale with a VLM-based mixture-of-transformers model in Chapter 3. Perhaps most importantly, we were able to study the properties of large-scale robot pre-training, and learned lasting lessons: e.g., the effectiveness of the diffusion objective, the superiority of the decoder-only transformer, and the correlation of performance with model size.

Scaling up model size immediately uncovered an issue that had received comparatively little attention beforehand: that of inference latency. Unlike digital AI agents, robot policies must act in real time; but like digital AI agents, we quickly found that larger models, now weighing in at billions of parameters, were necessary for state-of-the-art performance. Billion-parameter models introduce compounding sources of latency not just from increased compute and memory bandwidth requirements, but also network latency for robots without sufficient onboard processing power. In Part II, we found that naive execution strategies broke down in the presence of high latencies, leading to disastrous discontinuities in a robot's control stream. We then developed two algorithms to remedy this issue: an inference-time algorithm (Chapter 4) and a fine-tuning-time algorithm (Chapter 5), each with different advantages. Our algorithms significantly improved task completion throughput on a wide variety of difficult, cutting-edge manipulation tasks, and remained robust to latencies up to 300 milliseconds.

Finally, we explored a way to incorporate large-scale *non-robot* pre-training to directly improve

robot policies. Ultimately, the incredible capabilities of digital AI agents, powered by the mind-numbing quantity of Internet data, are too valuable to ignore; most likely, truly general robots will need to leverage both non-robot and robot data alike. In Part III, we found a way to decompose a robot policy into a high-level subgoal generator (i.e., world model) and a low-level controller, such that the high-level subgoal generator can easily take advantage of non-robot data: first, by initializing from widely available pre-trained image generation models, and second, by directly co-training on non-robot Internet video data. In Chapter 6, a small-scale version of this system outperformed a traditional flat policy with access to an order of magnitude more compute and robot data; one which also co-trained using non-robot Internet data, but in a more naive way. In Chapter 7, a larger-scale version of this system helped the robot perform a novel task, unseen in the robot training data; it also enabled zero-shot cross-embodiment transfer, i.e., performing a task on one robot embodiment using only demonstrations of that task from a different robot embodiment.

The journey to general-purpose robots is far from over. The research presented in this dissertation raises more questions than it answers — especially as the scale of robot datasets, models, and deployments continue to increase rapidly. We highlight a few promising and important future directions for research below.

Scaling laws. In Chapter 2, we conducted a basic scaling study, comparing the performance of identical model architectures at 10M, 27M, and 93M parameters. This result is but a weak imitation of the scaling laws that have been discovered in other domains, such as language modeling [127], which are far more comprehensive and robust. Robust scaling laws allow researchers to predict downstream task performance for a given model and dataset size *ahead of time*, which is critical for justifying the incredible financial and time commitment required to train a large frontier model. Numerous challenges make the science of scaling much more difficult in a robotics setting: the heterogeneity of data means that dataset composition and quality often has a greater effect than dataset size, and downstream task evaluations are much more expensive yet much more specific and may not cover the full range of use cases that a truly general robot should target. Nevertheless, we expect that some sort of scaling science must be established — not necessarily in the exact same way as in language modeling — to better guide large-scale robot data collection, model training, and evaluation.

Reinforcement learning. The informed reader will have immediately noticed the conspicuous absence of reinforcement learning (RL) from this dissertation. While RL has long been a popular research subject in the field of robotics, most recent advancements in general-purpose robot manipulation have come from imitation learning alone. However, there are many reasons to expect that to change. Early advances in natural language processing also mostly came from large-scale imitation — but those were soon followed by the application of RL techniques, such as RLHF [243] and RLVR [169, 71], that proved essential in creating the powerful AI agents we are familiar with today. In robotics, a vast majority of available data is clearly suboptimal, and that proportion will only increase as more autonomous robots are

deployed, each generating imperfect data from imperfect policies. It stands to reason that methods that can best leverage that suboptimal data will far outperform any others; whether through *offline* RL (e.g., reasoning about optimality during pre-training), or through *online* RL (e.g., enabling deployed policies to learn and improve in real-time).

Closed-loop control. All of the policies discussed in this dissertation use action chunking [375, 62], a pseudo-open-loop method where the robot generates a sequence of actions and executes the entire thing before re-planning. This means the robot's compute unit is *deliberately* idle some portion of the time. Action chunking has become the de facto standard, because naively removing it is catastrophic for policy performance in most manipulation tasks; yet, one might imagine that a method that can successfully utilize the robot's compute *all the time* would be able to more quickly ingest new information, producing more reactive and super-human policies. The algorithms presented in Part II merely smooth over inference latency — they do not address this deficiency. An extension of this work that enables maximally closed-loop control is a promising way to enable more dexterous and reactive robots.

World modeling. A *world model* is a recently popular catch-all term for models that predict future observations. Later research has already eclipsed the scale and scope of the image subgoal generators in Part III — e.g., policies based on much larger video generation models that are trained on far larger corpora of both robot and non-robot data [355, 239]. For the time being, non-robot video data is still cheaper and far more plentiful than robot data, and so methods that take advantage of it are promising. However, the details remain unsettled: what is the best way to mix robot and non-robot data? Is it strictly necessary to generate future observations at inference time, or is using them at training time sufficient? Can using a world model in a more sophisticated way, such as to plan in observation space, lead to even better performance?

Bibliography

[1] Josh Achiam et al. "Gpt-4 technical report". In: *arXiv preprint arXiv:2303.08774* (2023).

[2] AgiBot-World-Contributors et al. "AgiBot World Colosseo: A Large-scale Manipulation Platform for Scalable and Intelligent Embodied Systems". In: *arXiv preprint arXiv:2503.06669* (2025).

[3] Michael Ahn et al. "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances". In: *Conference on Robot Learning (CoRL)* (2022).

[4] Bo Ai et al. "Towards embodiment scaling laws in robot locomotion". In: *Conference on Robot Learning (CoRL)* (2025).

[5] Scale AI. *Introducing Scale's Automotive Foundation Model*. 2023. URL: `https://scale.com/blog/afm1`.

[6] Anurag Ajay et al. "Compositional Foundation Models for Hierarchical Planning". In: *arXiv preprint arXiv:2309.08587* (2023).

[7] Anurag Ajay et al. "Is Conditional Generative Modeling all you need for Decision Making?" In: *The Eleventh International Conference on Learning Representations*. 2023. URL: `https://openreview.net/forum?id=sP1fo2K9DFG`.

[8] Anurag Ajay et al. "Is conditional generative modeling all you need for decision-making?" In: *arXiv preprint arXiv:2211.15657* (2022).

[9] Jean-Baptiste Alayrac et al. "Flamingo: a visual language model for few-shot learning". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 23716–23736.

[10] Jorge Aldaco et al. "ALOHA 2: An Enhanced Low-Cost Hardware for Bimanual Teleoperation". In: *arXiv preprint arXiv:2405.02292* (2024).

[11] Ali Amin et al. "$\pi^*_{0.6}$: a VLA That Learns From Experience". In: *arXiv preprint arXiv:2511.14759* (2025).

[12] Marcin Andrychowicz et al. "Hindsight experience replay". In: *NeurIPS*. 2017.

[13] Mohammad Babaeizadeh et al. "FitVid: Overfitting in Pixel-Level Video Prediction". In: *arXiv preprint arXiv:2106.13195* (2020).

[14] Shikhar Bahl, Abhinav Gupta, and Deepak Pathak. "Human-to-robot imitation in the wild". In: *arXiv preprint arXiv:2207.09450* (2022).

[15] Shikhar Bahl et al. "Affordances from Human Videos as a Versatile Representation for Robotics". In: *CVPR*. 2023.

[16] Atilim Gunes Baydin et al. "Automatic differentiation in machine learning: a survey". In: *Journal of machine learning research* 18.153 (2018), pp. 1–43.

[17] Suneel Belkhale, Yuchen Cui, and Dorsa Sadigh. "HYDRA: Hybrid Robot Actions for Imitation Learning". In: *arxiv* (2023).

[18] Suneel Belkhale and Dorsa Sadigh. *MiniVLA: A Better VLA with a Smaller Footprint*. 2024. URL: `https://github.com/Stanford-ILIAD/openvla-mini`.

[19] Lucas Beyer et al. "PaliGemma: A versatile 3B VLM for transfer". In: *arXiv preprint arXiv:2407.07726* (2024).

[20] Homanga Bharadhwaj, Abhinav Gupta, and Shubham Tulsiani. "Visual affordance prediction for guiding robot exploration". In: *arXiv preprint arXiv:2305.17783* (2023).

[21] Homanga Bharadhwaj et al. "RoboAgent: Generalization and efficiency in robot manipulation via semantic augmentations and action chunking". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 4788–4795.

[22] Homanga Bharadhwaj et al. "Roboagent: Generalization and efficiency in robot manipulation via semantic augmentations and action chunking". In: *arXiv preprint arXiv:2309.01918* (2023).

[23] Homanga Bharadhwaj et al. "Track2act: Predicting point tracks from internet videos enables generalizable robot manipulation". In: *European Conference on Computer Vision*. Springer. 2024, pp. 306–324.

[24] Homanga Bharadhwaj et al. "Zero-shot robot manipulation from passive human videos". In: *arXiv preprint arXiv:2302.02011* (2023).

[25] Chethan Bhateja et al. "Robotic Offline RL from Internet Videos via Value-Function Pre-Training". In: 2023. URL: `https://api.semanticscholar.org/CorpusID:262217278`.

[26] Chethan Bhateja et al. "Robotic offline rl from internet videos via value-function pre-training". In: *arXiv preprint arXiv:2309.13041* (2023).

[27] Johan Bjorck et al. "GR00T N1: An Open Foundation Model for Generalist Humanoid Robots". In: *arXiv preprint arXiv:2503.14734* (2025).

[28] Johan Bjorck et al. "Gr00t n1: An open foundation model for generalist humanoid robots". In: *arXiv preprint arXiv:2503.14734* (2025).

[29] Kevin Black, Manuel Y Galliker, and Sergey Levine. "Real-Time Execution of Action Chunking Flow Policies". In: *arXiv preprint arXiv:2506.07339* (2025).

[30] Kevin Black, Manuel Y. Galliker, and Sergey Levine. "Real-Time Execution of Action Chunking Flow Policies". In: *Advances in Neural Information Processing Systems*. Vol. 38. Curran Associates, Inc., 2025, pp. 33383–33407. DOI: 10.52202/085713-1122. URL: https://proceedings.neurips.cc/paper_files/paper/2025/file/300ccb2187dedd4edcc07f7e76d8e553-Paper-Conference.pdf.

[31] Kevin Black et al. "$\pi_{0.5}$: a Vision-Language-Action Model with Open-World Generalization". In: *9th Annual Conference on Robot Learning*. 2025.

[32] Kevin Black et al. "π_0: A Vision-Language-Action Flow Model for General Robot Control". In: *arXiv preprint arXiv:2410.24164* (2024).

[33] Kevin Black et al. "π_0: A Vision-Language-Action Flow Model for General Robot Control". In: *arXiv preprint arXiv:2410.24164* (2024).

[34] Kevin Black et al. "Training-time action conditioning for efficient real-time chunking". In: *arXiv preprint arXiv:2512.05964* (2025).

[35] Kevin Black et al. "Zero-Shot Robotic Manipulation with Pre-Trained Image-Editing Diffusion Models". In: *International Conference on Learning Representations*. Vol. 2024. 2024, pp. 33431–33452. URL: https://proceedings.iclr.cc/paper_files/paper/2024/file/8ea50bf458f6070548b11babbe0bf89b-Paper-Conference.pdf.

[36] Mariusz Bojarski et al. "End to end learning for self-driving cars". In: *arXiv preprint arXiv:1604.07316* (2016).

[37] Konstantinos Bousmalis et al. "RoboCat: A Self-Improving Foundation Agent for Robotic Manipulation". In: *arXiv preprint arXiv:2306.11706* (2023).

[38] Max Braun et al. "Riemannian flow matching policy for robot motion learning". In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2024, pp. 5144–5151.

[39] Greg Brockman et al. *Introducing ChatGPT and Whisper APIs*. OpenAI Blog. 2023. URL: https://openai.com/blog/introducing-chatgpt-and-whisper-apis.

[40] Anthony Brohan et al. "Do as i can, not as i say: Grounding language in robotic affordances". In: *Conference on Robot Learning*. PMLR. 2023, pp. 287–318.

[41] Anthony Brohan et al. "Do as i can, not as i say: Grounding language in robotic affordances". In: *Conference on Robot Learning*. PMLR. 2023, pp. 287–318.

[42] Anthony Brohan et al. "Rt-1: Robotics transformer for real-world control at scale". In: *arXiv preprint arXiv:2212.06817* (2022).

[43] Anthony Brohan et al. "RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control". In: *arXiv preprint arXiv:2307.15818*. 2023.

[44] Anthony Brohan et al. "Rt-2: Vision-language-action models transfer web knowledge to robotic control". In: *arXiv preprint arXiv:2307.15818* (2023).

[45] Anthony Brohan et al. "Rt-2: Vision-language-action models transfer web knowledge to robotic control". In: *arXiv preprint arXiv:2307.15818* (2023).

[46] Anthony Brohan et al. "Rt-2: Vision-language-action models transfer web knowledge to robotic control". In: *arXiv preprint arXiv:2307.15818* (2023).

[47] 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*. 2023, pp. 18392–18402.

[48] Serkan Cabi et al. "Scaling data-driven robotics with reward sketching and batch reinforcement learning". In: *arXiv preprint arXiv:1909.12200* (2019).

[49] Serkan Cabi et al. "Scaling data-driven robotics with reward sketching and batch reinforcement learning". In: *arXiv preprint arXiv:1909.12200* (2019).

[50] Holger Caesar et al. "nuscenes: A multimodal dataset for autonomous driving". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11621–11631.

[51] Chi-Lam Cheang et al. "GR-2: A Generative Video-Language-Action Model with Web-Scale Knowledge for Robot Manipulation". In: *arXiv preprint arXiv:2410.06158* (2024).

[52] Chi-Lam Cheang et al. "GR-2: A Generative Video-Language-Action Model with Web-Scale Knowledge for Robot Manipulation". In: *arXiv preprint arXiv:2410.06158* (2024).

[53] Yevgen Chebotar et al. "Actionable Models: Unsupervised Offline Reinforcement Learning of Robotic Skills". In: *arXiv preprint arXiv:2104.07749* (2021).

[54] Hongyi Chen et al. "Dexterous manipulation policies from rgb human videos via 3d hand-object trajectory reconstruction". In: *arXiv preprint arXiv:2602.09013* (2026).

[55] Lawrence Yunliang Chen, Simeon Adebola, and Ken Goldberg. *Berkeley UR5 Demonstration Dataset*. `https://sites.google.com/view/berkeley-ur5/home`.

[56] Lili Chen et al. "Decision transformer: Reinforcement learning via sequence modeling". In: *arXiv preprint arXiv:2106.01345* (2021).

[57] William Chen et al. "Vision-Language Models Provide Promptable Representations for Reinforcement Learning". In: *arXiv preprint arXiv:2402.02651* (2024).

[58] Zoey Chen et al. "Genaug: Retargeting behaviors to unseen situations via generative augmentation". In: *arXiv preprint arXiv:2302.06671* (2023).

[59] Zoey Chen et al. "Genaug: Retargeting behaviors to unseen situations via generative augmentation". In: *arXiv preprint arXiv:2302.06671* (2023).

[60] An-Chieh Cheng et al. "NaVILA: Legged Robot Vision-Language-Action Model for Navigation". In: *arXiv preprint arXiv:2412.04453* (2024).

[61] Cheng Chi et al. "Diffusion Policy: Visuomotor Policy Learning via Action Diffusion". In: *Proceedings of Robotics: Science and Systems (RSS)*. 2023.

[62] Cheng Chi et al. "Diffusion policy: Visuomotor policy learning via action diffusion". In: *The International Journal of Robotics Research* (2023), p. 02783649241273668.

[63] Cheng Chi et al. "Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots". In: *arXiv preprint arXiv:2402.10329* (2024).

[64] Hyung Won Chung et al. "Scaling instruction-finetuned language models". In: *arXiv preprint arXiv:2210.11416* (2022).

[65] OX-Embodiment Collaboration et al. "Open X-Embodiment: Robotic learning datasets and RT-X models". In: *arXiv preprint arXiv:2310.08864* 1.2 (2023).

[66] Open X-Embodiment Collaboration et al. *Open X-Embodiment: Robotic Learning Datasets and RT-X Models*. `https://robotics-transformer-x.github.io`. 2023.

[67] Donald R. Coughanowr and Steven E. LeBlanc. "Process Systems Analysis and Control". In: 3rd. New York: McGraw-Hill, 2009. Chap. 18. ISBN: 978-0073397894.

[68] Zichen Jeff Cui et al. "From Play to Policy: Conditional Behavior Generation from Uncurated Robot Data". In: *The Eleventh International Conference on Learning Representations*. 2022.

[69] Sudeep Dasari et al. "RoboNet: Large-Scale Multi-Robot Learning". In: *Conference on Robot Learning*. PMLR. 2020, pp. 885–897.

[70] Shivin Dass et al. *CLVR Jaco Play Dataset*. Version 1.0.0. 2023. URL: `https://github.com/clvrai/clvr_jaco_play_dataset`.

[71] DeepSeek-AI. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. 2025. arXiv: `2501.12948 [cs.CL]`. URL: `https://arxiv.org/abs/2501.12948`.

[72] Chaorui Deng et al. "Emerging properties in unified multimodal pretraining". In: *arXiv preprint arXiv:2505.14683* (2025).

[73] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[74] Ria Doshi et al. "Scaling Cross-Embodied Learning: One Policy for Manipulation, Navigation, Locomotion and Aviation". In: *Conference on Robot Learning*. 2024.

[75] Alexey Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[76] Danny Driess, Fei Xia, Mehdi S.M. Sajjadi, et al. "PaLM-E: An Embodied Multimodal Language Model". In: *International Conference on Machine Learning (ICML)* (2023).

[77] Danny Driess et al. "Knowledge insulating vision-language-action models: Train fast, run fast, generalize better". In: *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS)*. 2025.

[78] Danny Driess et al. "Palm-e: An embodied multimodal language model". In: *arXiv preprint arXiv:2303.03378* (2023).

[79] Danny Driess et al. "Palm-e: An embodied multimodal language model". In: *arXiv preprint arXiv:2303.03378* (2023).

[80] Maximilian Du et al. "Behavior Retrieval: Few-Shot Imitation Learning by Querying Unlabeled Datasets". In: *ArXiv* abs/2304.08742 (2023). URL: `https://api.semanticscholar.org/CorpusID:258186973`.

[81] Nan Du et al. "Glam: Efficient scaling of language models with mixture-of-experts". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 5547–5569.

[82] Yilun Du et al. "Learning Universal Policies via Text-Guided Video Generation". In: *arXiv preprint arXiv:2302.00111* (2023).

[83] Yilun Du et al. "Learning Universal Policies via Text-Guided Video Generation". In: *Advances in Neural Information Processing Systems (NeurIPS)* (2023).

[84] Yilun Du et al. "Video Language Planning". In: *International Conference on Learning Representations (ICLR)* (2024).

[85] Frederik Ebert et al. "Bridge data: Boosting generalization of robotic skills with cross-domain datasets". In: *arXiv preprint arXiv:2109.13396* (2021).

[86] Frederik Ebert et al. "Bridge data: Boosting generalization of robotic skills with cross-domain datasets". In: *arXiv preprint arXiv:2109.13396* (2021).

[87] Frederik Ebert et al. "Visual foresight: Model-based deep reinforcement learning for vision-based robotic control". In: *arXiv preprint arXiv:1812.00568* (2018).

[88] Patrick Esser et al. "Scaling rectified flow transformers for high-resolution image synthesis". In: *Forty-first International Conference on Machine Learning*. 2024.

[89] Haritheja Etukuru et al. "Robot utility models: General policies for zero-shot deployment in new environments". In: *arXiv preprint arXiv:2409.05865* (2024).

[90] Hao-Shu Fang et al. "Rh20t: A comprehensive robotic dataset for learning diverse skills in one-shot". In: *Towards Generalist Robots: Learning Paradigms for Scalable Skill Acquisition@ CoRL2023* 3 (2023), p. 5.

[91] Hao-Shu Fang et al. "Rh20t: A comprehensive robotic dataset for learning diverse skills in one-shot". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 653–660.

[92] Haoquan Fang et al. "Sam2act: Integrating visual foundation model with a memory architecture for robotic manipulation". In: *arXiv preprint arXiv:2501.18564* (2025).

[93] William Fedus, Barret Zoph, and Noam Shazeer. "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity". In: *Journal of Machine Learning Research* 23.120 (2022), pp. 1–39.

[94] Chelsea Finn and Sergey Levine. "Deep visual foresight for planning robot motion". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 2786–2793.

[95] Vlad Firoiu, Tina Ju, and Josh Tenenbaum. "At human speed: Deep reinforcement learning with action delay". In: *arXiv preprint arXiv:1810.07286* (2018).

[96] Kevin Frans et al. "One step diffusion via shortcut models". In: *arXiv preprint arXiv:2410.12557* (2024).

[97] Zipeng Fu, Tony Z Zhao, and Chelsea Finn. "Mobile ALOHA: Learning Bimanual Mobile Manipulation with Low-Cost Whole-Body Teleoperation". In: *arXiv preprint arXiv:2401.02117* (2024).

[98] Ruiqi Gao et al. "Diffusion Meets Flow Matching: Two Sides of the Same Coin". In: 2024. URL: `https://diffusionflow.github.io/`.

[99] Yuying Ge et al. "Policy Adaptation From Foundation Model Feedback". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 19059–19069.

[100] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for autonomous driving? the kitti vision benchmark suite". In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE. 2012, pp. 3354–3361.

[101] Abraham George and Amir Barati Farimani. "One act play: Single demonstration behavior cloning with action chunking transformers". In: *arXiv preprint arXiv:2309.10175* (2023).

[102] Raghav Goyal et al. "The" something something" video database for learning and evaluating visual common sense". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 5842–5850.

[103] Kristen Grauman et al. "Ego4d: Around the world in 3,000 hours of egocentric video". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 18995–19012.

[104] Shresth Grover et al. "Enhancing generalization in vision-language-action models by preserving pretrained representations". In: *arXiv preprint arXiv:2509.11417* (2025).

[105] Jiayuan Gu et al. "RT-Trajectory: Robotic Task Generalization via Hindsight Trajectory Sketches". In: *International Conference on Learning Representations (ICLR)* (2024).

[106] Abhinav Gupta et al. "Robot learning in homes: Improving generalization and reducing dataset bias". In: *Advances in neural information processing systems* 31 (2018).

[107] Abhinav Gupta et al. "Robot learning in homes: Improving generalization and reducing dataset bias". In: *Advances in neural information processing systems* 31 (2018).

[108] Anthony Ha. *Google's Gemini has beaten Pokémon Blue (with a little help)*. `https://techcrunch.com/2025/05/03/googles-gemini-has-beaten-pokemon-blue-with-a-little-help/`. Accessed May 8, 2025. May 2025.

[109] Huy Ha, Pete Florence, and Shuran Song. "Scaling up and distilling down: Language-guided robot skill acquisition". In: *Conference on Robot Learning*. PMLR. 2023, pp. 3766–3777.

[110] Pim de Haan, Dinesh Jayaraman, and Sergey Levine. "Causal confusion in imitation learning". In: *NeurIPS* (2019).

[111] Danijar Hafner et al. "Dream to control: Learning behaviors by latent imagination". In: *arXiv preprint arXiv:1912.01603* (2019).

[112] Danijar Hafner et al. "Mastering diverse domains through world models". In: *arXiv preprint arXiv:2301.04104* (2023).

[113] Nicklas Hansen, Xiaolong Wang, and Hao Su. *Temporal Difference Learning for Model Predictive Control*. 2022. arXiv: `2203.04955 [cs.LG]`. URL: `https://arxiv.org/abs/2203.04955`.

[114] Philippe Hansen-Estruch et al. "IDQL: Implicit Q-Learning as an Actor-Critic Method with Diffusion Policies". In: *arXiv preprint arXiv:2304.10573* (2023).

[115] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[116] Wanggui He et al. "Mars: Mixture of auto-regressive models for fine-grained text-to-image synthesis". In: *arXiv preprint arXiv:2407.07614* (2024).

[117] Zihao He et al. "Scaling Cross-Embodiment World Models for Dexterous Manipulation". In: *arXiv preprint arXiv:2511.01177* (2025).

[118] Joey Hejna et al. "Robot data curation with mutual information estimators". In: *arXiv preprint arXiv:2502.08623* (2025).

[119] Minho Heo et al. "FurnitureBench: Reproducible Real-World Benchmark for Long-Horizon Complex Manipulation". In: *Robotics: Science and Systems*. 2023.

[120] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models". In: *Advances in neural information processing systems* 33 (2020), pp. 6840–6851.

[121] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models". In: *Advances in neural information processing systems* 33 (2020), pp. 6840–6851.

[122] Jonathan Ho and Tim Salimans. "Classifier-free diffusion guidance". In: *arXiv preprint arXiv:2207.12598* (2022).

[123] Jonathan Ho and Tim Salimans. "Classifier-free diffusion guidance". In: *arXiv preprint arXiv:2207.12598* (2022).

[124] Jonathan Ho et al. *Imagen Video: High Definition Video Generation with Diffusion Models*. 2022. arXiv: `2210.02303 [cs.CV]`.

[125] Jonathan Ho et al. "Video diffusion models". In: *arXiv:2204.03458* (2022).

[126] Sigmund H Høeg, Yilun Du, and Olav Egeland. "Streaming Diffusion Policy: Fast Policy Synthesis with Variable Noise Diffusion Models". In: *arXiv preprint arXiv:2406.04806* (2024).

[127] Jordan Hoffmann et al. *Training Compute-Optimal Large Language Models*. 2022. arXiv: 2203.15556 [cs.CL]. URL: https://arxiv.org/abs/2203.15556.

[128] Anthony Hu et al. *GAIA-1: A Generative World Model for Autonomous Driving*. 2023. arXiv: 2309.17080 [cs.CV].

[129] Chenguang Huang et al. "Audio Visual Language Maps for Robot Navigation". In: *Proceedings of the International Symposium on Experimental Robotics (ISER)*. Chiang Mai, Thailand, 2023.

[130] Chenguang Huang et al. "Visual language maps for robot navigation". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 10608–10615.

[131] Wenlong Huang et al. "Inner monologue: Embodied reasoning through planning with language models". In: *arXiv preprint arXiv:2207.05608* (2022).

[132] Wenlong Huang et al. "Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents". In: *arXiv preprint arXiv:2201.07207* (2022).

[133] Wenlong Huang et al. "Voxposer: Composable 3d value maps for robotic manipulation with language models". In: *arXiv preprint arXiv:2307.05973* (2023).

[134] Physical Intelligence et al. "$\pi_{0.5}$: A Vision-Language-Action Model with Open-World Generalization". In: *arXiv preprint arXiv:2504.16054* (2025).

[135] Benoit Jacob et al. "Quantization and training of neural networks for efficient integer-arithmetic-only inference". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2704–2713.

[136] Eric Jang et al. "Bc-z: Zero-shot task generalization with robotic imitation learning". In: *Conference on Robot Learning*. PMLR. 2022, pp. 991–1002.

[137] Huiwon Jang et al. "ContextVLA: Vision-Language-Action Model with Amortized Multi-Frame Context". In: *arXiv preprint arXiv:2510.04246* (2025).

[138] Michael Janner, Qiyang Li, and Sergey Levine. "Offline Reinforcement Learning as One Big Sequence Modeling Problem". In: *Advances in Neural Information Processing Systems*. 2021.

[139] Michael Janner et al. "Planning with Diffusion for Flexible Behavior Synthesis". In: *International Conference on Machine Learning*. 2022.

[140] Michael Janner et al. "Planning with diffusion for flexible behavior synthesis". In: *arXiv preprint arXiv:2205.09991* (2022).

[141] Tao Jiang et al. "Galaxea open-world dataset and g0 dual-system vla model". In: *arXiv preprint arXiv:2509.00576* (2025).

[142] Yunfan Jiang et al. "VIMA: General Robot Manipulation with Multimodal Prompts". In: *International Conference on Machine Learning (ICML)* (2023).

[143] Yunfan Jiang et al. "VIMA: Robot Manipulation with Multimodal Prompts". In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by Andreas Krause et al. Vol. 202. Proceedings of Machine Learning Research. PMLR, 23–29 Jul 2023, pp. 14975–15022. URL: https://proceedings.mlr.press/v202/jiang23b.html.

[144] Zhenyu Jiang et al. "DexMimicGen: Automated Data Generation for Bimanual Dexterous Manipulation via Imitation Learning". In: *arXiv preprint arXiv:2410.24185* (2024).

[145] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (2021), pp. 583–589.

[146] Dmitry Kalashnikov et al. "QT-Opt: Scalable deep reinforcement learning for vision-based robotic manipulation". In: *arXiv preprint arXiv:1806.10293* (2018).

[147] Dmitry Kalashnikov et al. "Scalable deep reinforcement learning for vision-based robotic manipulation". In: *Conference on robot learning*. PMLR. 2018, pp. 651–673.

[148] Dmitry Kalashnikov et al. "Scaling up multi-task robotic reinforcement learning". In: *5th Annual Conference on Robot Learning*. 2021.

[149] Aishwarya Kamath et al. "Mdetr-modulated detection for end-to-end multi-modal understanding". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 1780–1790.

[150] Ivan Kapelyukh, Vitalis Vosylius, and Edward Johns. "DALL-E-Bot: Introducing Web-Scale Diffusion Models to Robotics". In: (2023).

[151] Ivan Kapelyukh, Vitalis Vosylius, and Edward Johns. "Dall-e-bot: Introducing web-scale diffusion models to robotics". In: *IEEE Robotics and Automation Letters* 8.7 (2023), pp. 3956–3963.

[152] Jared Kaplan et al. "Scaling laws for neural language models". In: *arXiv preprint arXiv:2001.08361* (2020).

[153] Siddharth Karamcheti et al. "Language-driven representation learning for robotics". In: *arXiv preprint arXiv:2302.12766* (2023).

[154] Simar Kareer et al. "Egomimic: Scaling imitation learning via egocentric video". In: *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2025, pp. 13226–13233.

[155] Simar Kareer et al. *Emergence of Human to Robot Transfer in Vision-Language-Action Models*. 2025. arXiv: 2512.22414 [cs.RO]. URL: https://arxiv.org/abs/2512.22414.

[156] Haresh Karnan et al. "Socially compliant navigation dataset (scand): A large-scale dataset of demonstrations for social navigation". In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 11807–11814.

[157] Alexander Khazatsky et al. "DROID: A large-scale in-the-wild robot manipulation dataset". In: *arXiv preprint arXiv:2403.12945* (2024).

[158] Hanjung Kim et al. "UniSkill: Imitating Human Videos via Cross-Embodiment Skill Representations". In: *Conference on Robot Learning*. PMLR. 2025, pp. 4269–4294.

[159] Moo Jin Kim, Chelsea Finn, and Percy Liang. "Fine-Tuning Vision-Language-Action Models: Optimizing Speed and Success". In: *arXiv preprint arXiv:2502.19645* (2025).

[160] Moo Jin Kim et al. "Cosmos Policy: Fine-Tuning Video Models for Visuomotor Control and Planning". In: *International Conference on Learning Representations (ICLR)*. 2026.

[161] Moo Jin Kim et al. "OpenVLA: An Open-Source Vision-Language-Action Model". In: *arXiv preprint arXiv:2406.09246* (2024).

[162] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *International Conference on Learning Representations (ICLR)* (2015).

[163] Alexander Kirillov et al. *Segment Anything*. Apr. 2023. DOI: 10.48550/arXiv.2304.02643. arXiv: 2304.02643 [cs]. (Visited on 12/10/2023).

[164] Po-Chen Ko et al. "Learning to Act from Actionless Videos through Dense Correspondences". In: *The Twelfth International Conference on Learning Representations*. 2024.

[165] Vikash Kumar et al. "RoboHive: A Unified Framework for Robot Learning". In: *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. 2023. URL: https://openreview.net/forum?id=0H5fRQcpQ7.

[166] Teyun Kwon, Norman Di Palo, and Edward Johns. "Language Models as Zero-Shot Trajectory Generators". In: *arXiv preprint arXiv:2310.11604* (2023).

[167] Woosuk Kwon et al. "Efficient memory management for large language model serving with pagedattention". In: *Proceedings of the 29th Symposium on Operating Systems Principles*. 2023, pp. 611–626.

[168] Lucy Lai, Ann Zixiang Huang, and Samuel J Gershman. "Action chunking as policy compression". In: (2022).

[169] Nathan Lambert et al. *Tülu 3: Pushing Frontiers in Open Language Model Post-Training*. 2024. arXiv: 2411.15124 [cs.CL]. URL: https://arxiv.org/abs/2411.15124.

[170] Alex X Lee et al. "Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 741–752.

[171] Donghwan Lee and Niao He. "Stochastic primal-dual Q-learning". In: *arXiv preprint arXiv:1810.08298* (2018).

[172] Seungjae Lee et al. "Behavior generation with latent actions". In: *arXiv preprint arXiv:2403.03181* (2024).

[173] Dmitry Lepikhin et al. "Gshard: Scaling giant models with conditional computation and automatic sharding". In: *arXiv preprint arXiv:2006.16668* (2020).

[174] Sergey Levine et al. "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[175] Sergey Levine et al. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *The International journal of robotics research* 37.4-5 (2018), pp. 421–436.

[176] Sergey Levine et al. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *The International journal of robotics research* 37.4-5 (2018), pp. 421–436.

[177] Hao Li et al. "CronusVLA: Transferring Latent Motion Across Time for Multi-Frame Prediction in Manipulation". In: *arXiv preprint arXiv:2506.19816* (2025).

[178] Qixiu Li et al. "Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation". In: *arXiv preprint arXiv:2411.19650* (2024).

[179] Xinghang Li et al. "Vision-Language Foundation Models as Effective Robot Imitators". In: *International Conference on Learning Representations (ICLR)* (2024).

[180] Yujia Li et al. "Competition-level code generation with AlphaCode". In: *Science* 378.6624 (2022), pp. 1092–1097.

[181] Yunfei Li et al. "Gr-rl: Going dexterous and precise for long-horizon robotic manipulation". In: *arXiv preprint arXiv:2512.01801* (2025).

[182] Zuolei Li et al. *LatBot: Distilling Universal Latent Actions for Vision-Language-Action Models*. 2025. arXiv: `2511.23034 [cs.RO]`. URL: `https://arxiv.org/abs/2511.23034`.

[183] Jacky Liang et al. "Code as Policies: Language Model Programs for Embodied Control". In: *IEEE International Conference on Robotics and Automation (ICRA)* (2023).

[184] Jacky Liang et al. "Code as policies: Language model programs for embodied control". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 9493–9500.

[185] Junbang Liang et al. "Dreamitate: Real-World Visuomotor Policy Learning via Video Generation". In: *Conference on Robot Learning*. PMLR. 2025, pp. 3943–3960.

[186] Junbang Liang et al. "Video generators are robot policies". In: *arXiv preprint arXiv:2508.00795* (2025).

[187] Weixin Liang et al. "Mixture-of-transformers: A sparse and scalable architecture for multi-modal foundation models". In: *arXiv preprint arXiv:2411.04996* (2024).

[188] Fanqi Lin et al. "Onetwovla: A unified vision-language-action model with adaptive reasoning". In: *arXiv preprint arXiv:2505.11917* (2025).

[189] Ji Lin et al. "Awq: Activation-aware weight quantization for on-device llm compression and acceleration". In: *Proceedings of Machine Learning and Systems* 6 (2024), pp. 87–100.

[190] Xiaopeng Lin et al. *PhysBrain: Human Egocentric Data as a Bridge from Vision Language Models to Physical Intelligence*. 2025. arXiv: 2512.16793 [cs.RO]. URL: https://arxiv.org/abs/2512.16793.

[191] Yixin Lin et al. *Polymetis*. https://facebookresearch.github.io/fairo/polymetis/. 2021.

[192] Yaron Lipman et al. "Flow matching for generative modeling". In: *arXiv preprint arXiv:2210.02747* (2022).

[193] Bingchen Liu et al. "Playground v3: Improving text-to-image alignment with deep-fusion large language models". In: *arXiv preprint arXiv:2409.10695* (2024).

[194] Haotian Liu et al. "Visual instruction tuning". In: *Advances in neural information processing systems* 36 (2024).

[195] Huihan Liu et al. "Robot Learning on the Job: Human-in-the-Loop Autonomy and Learning During Deployment". In: *Robotics: Science and Systems (RSS)*. 2023.

[196] Peiqi Liu et al. "Ok-robot: What really matters in integrating open-knowledge models for robotics". In: *arXiv preprint arXiv:2401.12202* (2024).

[197] Qiang Liu. "Rectified flow: A marginal preserving approach to optimal transport". In: *arXiv preprint arXiv:2209.14577* (2022).

[198] Songming Liu et al. "RDT-1B: a Diffusion Foundation Model for Bimanual Manipulation". In: *International Conference on Learning Representations (ICLR)* (2025).

[199] Songming Liu et al. "Rdt-1b: a diffusion foundation model for bimanual manipulation". In: *arXiv preprint arXiv:2410.07864* (2024).

[200] Xingchao Liu, Chengyue Gong, and Qiang Liu. "Flow straight and fast: Learning to generate and transfer data with rectified flow". In: *arXiv preprint arXiv:2209.03003* (2022).

[201] Yuejiang Liu et al. "Bidirectional Decoding: Improving Action Chunking via Closed-Loop Resampling". In: *arXiv preprint arXiv:2408.17355* (2024).

[202] Ilya Loshchilov and Frank Hutter. "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations*. 2018.

[203] Andreas Lugmayr et al. "Repaint: Inpainting using denoising diffusion probabilistic models". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 11461–11471.

[204] Hao Luo et al. *Being-H0.5: Scaling Human-Centric Robot Learning for Cross-Embodiment Generalization*. 2026. arXiv: 2601.12993 [cs.RO]. URL: https://arxiv.org/abs/2601.12993.

[205] Jianlan Luo et al. "FMB: a Functional Manipulation Benchmark for Generalizable Robotic Learning". In: *arXiv preprint arXiv:2401.08553* (2024).

[206] Jianlan Luo et al. "Multi-Stage Cable Routing through Hierarchical Imitation Learning". In: *arXiv preprint arXiv:2307.08927* (2023).

[207] Corey Lynch and Pierre Sermanet. "Language Conditioned Imitation Learning Over Unstructured Data". In: *RSS*. 2021.

[208] Corey Lynch et al. "Interactive language: Talking to robots in real time". In: *IEEE Robotics and Automation Letters* (2023).

[209] Yecheng Jason Ma et al. "LIV: Language-Image Representations and Rewards for Robotic Control". In: *arXiv preprint arXiv:2306.00958* (2023).

[210] Yecheng Jason Ma et al. "VIP: Towards Universal Visual Reward and Representation via Value-Implicit Pre-Training". In: *The Eleventh International Conference on Learning Representations*. 2022.

[211] Arjun Majumdar et al. "Where are we in the search for an Artificial Visual Cortex for Embodied Intelligence?" In: 2023. arXiv: 2303.18240 [cs.CV].

[212] Arjun Majumdar et al. "Where are we in the search for an Artificial Visual Cortex for Embodied Intelligence?" In: *arXiv preprint arXiv:2303.18240* (2023).

[213] Zhao Mandi et al. "Cacti: A framework for scalable multi-task multi-scene visual imitation learning". In: *arXiv preprint arXiv:2212.05711* (2022).

[214] Zhao Mandi et al. "Cacti: A framework for scalable multi-task multi-scene visual imitation learning". In: *arXiv preprint arXiv:2212.05711* (2022).

[215] Ajay Mandlekar et al. "MimicGen: A Data Generation System for Scalable Robot Learning using Human Demonstrations". In: *7th Annual Conference on Robot Learning*. 2023.

[216] Ajay Mandlekar et al. "MimicGen: A data generation system for scalable robot learning using human demonstrations". In: *arXiv preprint arXiv:2310.17596* (2023).

[217] Ajay Mandlekar et al. "RoboTurk: A Crowdsourcing Platform for Robotic Skill Learning through Imitation". In: *CoRR* abs/1811.02790 (2018). arXiv: 1811.02790. URL: http://arxiv.org/abs/1811.02790.

[218] Ajay Mandlekar et al. "RoboTurk: A crowdsourcing platform for robotic skill learning through imitation". In: *Conference on Robot Learning*. PMLR. 2018, pp. 879–893.

[219] Ajay Mandlekar et al. "Roboturk: A crowdsourcing platform for robotic skill learning through imitation". In: *Conference on Robot Learning*. PMLR. 2018, pp. 879–893.

[220] Morteza Mardani et al. "A variational perspective on solving inverse problems with diffusion models". In: *arXiv preprint arXiv:2305.04391* (2023).

[221] Michael Matthews et al. "Kinetix: Investigating the Training of General Agents through Open-Ended Physics-Based Control Tasks". In: *arXiv preprint arXiv:2410.23208* (2024).

[222] Oier Mees, Jessica Borja-Diaz, and Wolfram Burgard. "Grounding Language with Visual Affordances over Unstructured Data". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. London, UK, 2023.

[223] Oier Mees, Lukas Hermann, and Wolfram Burgard. "What matters in language conditioned robotic imitation learning over unstructured data". In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 11205–11212.

[224] Oier Mees, Lukas Hermann, and Wolfram Burgard. "What matters in language conditioned robotic imitation learning over unstructured data". In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 11205–11212.

[225] Oier Mees et al. "CALVIN: A Benchmark for Language-Conditioned Policy Learning for Long-Horizon Robot Manipulation Tasks". In: *IEEE Robotics and Automation Letters (RA-L)* 7.3 (2022), pp. 7327–7334.

[226] Oier Mees et al. "CALVIN: A Benchmark for Language-Conditioned Policy Learning for Long-Horizon Robot Manipulation Tasks". In: *IEEE Robotics and Automation Letters (RA-L)* 7.3 (2022), pp. 7327–7334.

[227] Russell Mendonca, Shikhar Bahl, and Deepak Pathak. "Structured World Models from Human Videos". In: *CoRL* (2023).

[228] Atharva Mete et al. *QueST: Self-Supervised Skill Abstractions for Learning Continuous Control*. 2024. arXiv: 2407.15840 [cs.RO]. URL: https://arxiv.org/abs/2407.15840.

[229] Matthias Minderer et al. "Simple open-vocabulary object detection". In: *European Conference on Computer Vision*. Springer. 2022, pp. 728–755.

[230] Vivek Myers et al. "Goal Representations for Instruction Following: A Semi-Supervised Language Interface to Control". In: *Conference on Robot Learning (CoRL)* (2023).

[231] Vivek Myers et al. "Goal Representations for Instruction Following: A Semi-Supervised Language Interface to Control". In: *arXiv preprint arXiv:2307.00117* (2023).

[232] Ashvin V Nair et al. "Visual reinforcement learning with imagined goals". In: *Advances in neural information processing systems* 31 (2018).

[233] Suraj Nair and Chelsea Finn. "Hierarchical Foresight: Self-Supervised Learning of Long-Horizon Tasks via Visual Subgoal Generation". In: *International Conference on Learning Representations*. 2020.

[234] Suraj Nair and Chelsea Finn. "Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation". In: *arXiv preprint arXiv:1909.05829* (2019).

[235] Suraj Nair et al. "R3M: A Universal Visual Representation for Robot Manipulation". In: *ArXiv* abs/2203.12601 (2022).

[236] Suraj Nair et al. "R3M: A Universal Visual Representation for Robot Manipulation". In: *Conference on Robot Learning*. PMLR. 2023, pp. 892–909.

[237] Soroush Nasiriany et al. "Learning and Retrieval from Prior Data for Skill-based Imitation Learning". In: *Conference on Robot Learning (CoRL)*. 2022.

[238] Alexander Quinn Nichol and Prafulla Dhariwal. "Improved denoising diffusion probabilistic models". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8162–8171.

[239] NVIDIA. "Cosmos 3: Omnimodal World Models for Physical AI". In: *arXiv preprint arXiv:2606.02800* (2026). URL: https://research.nvidia.com/labs/cosmos-lab/cosmos3/technical-report.pdf.

[240] Open X-Embodiment Collaboration et al. *Open X-Embodiment: Robotic Learning Datasets and RT-X Models*. https://arxiv.org/abs/2310.08864. 2023.

[241] OpenAI. *GPT-4 Technical Report*. Mar. 2023. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774 [cs]. (Visited on 12/10/2023).

[242] OpenAI. *Introducing OpenAI Codex*. Accessed on May 27, 2025. Aug. 2021. URL: https://openai.com/index/introducing-codex/.

[243] Long Ouyang et al. *Training Language Models to Follow Instructions with Human Feedback*. 2022. arXiv: 2203.02155 [cs.CL]. URL: https://arxiv.org/abs/2203.02155.

[244] Long Ouyang et al. "Training language models to follow instructions with human feedback". In: *Advances in neural information processing systems* 35 (2022), pp. 27730–27744.

[245] Jonas Pai et al. *mimic-video: Video-Action Models for Generalizable Robot Control Beyond VLAs*. 2025. arXiv: 2512.15692 [cs.RO]. URL: https://arxiv.org/abs/2512.15692.

[246] Jyothish Pari et al. *The Surprising Effectiveness of Representation Learning for Visual Imitation*. 2021. arXiv: 2112.01511 [cs.RO].

[247] Alexander Pashevich et al. "Learning to augment synthetic images for sim2real policy transfer". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 2651–2657.

[248] Deepak Pathak et al. "Zero-shot visual imitation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2018, pp. 2050–2053.

[249] Tim Pearce et al. "Imitating human behaviour with diffusion models". In: *arXiv preprint arXiv:2301.10677* (2023).

[250] William Peebles and Saining Xie. "Scalable diffusion models with transformers". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 4195–4205.

[251] Ethan Perez et al. *FiLM: Visual Reasoning with a General Conditioning Layer*. Dec. 2017. arXiv: 1709.07871 [cs, stat]. (Visited on 04/28/2023).

[252] Ethan Perez et al. "Film: Visual reasoning with a general conditioning layer". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[253] Karl Pertsch et al. "Fast: Efficient action tokenization for vision-language-action models". In: *arXiv preprint arXiv:2501.09747* (2025).

[254] Physical Intelligence et al. "$\pi_{0.7}$: a Steerable Generalist Robotic Foundation Model with Emergent Capabilities". In: *arXiv preprint arXiv:2604.15483* (2026). DOI: 10.48550/arXiv.2604.15483.

[255] Lerrel Pinto and Abhinav Gupta. "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours". In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 3406–3413.

[256] Lerrel Pinto and Abhinav Gupta. "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours". In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 3406–3413.

[257] Ashwini Pokle et al. "Training-free linear image inverses via flows". In: *arXiv preprint arXiv:2310.04432* (2023).

[258] Adam Polyak et al. "Movie Gen: A Cast of Media Foundation Models". In: *arXiv preprint arXiv:2410.13720* (2024).

[259] Aaditya Prasad et al. "Consistency policy: Accelerated visuomotor policies via consistency distillation". In: *arXiv preprint arXiv:2405.07503* (2024).

[260] Delin Qu et al. "SpatialVLA: Exploring Spatial Representations for Visual-Language-Action Model". In: *Robotics: Science and Systems (RSS)* (2025).

[261] Gabriel Quere et al. "Shared Control Templates for Assistive Robotics". en. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. Paris, France, 2020, p. 7.

[262] Alec Radford et al. *Language Models are Unsupervised Multitask Learners*. Tech. rep. OpenAI, 2019. URL: https://cdn.openai.com/better-language-models/language-models.pdf.

[263] Alec Radford et al. "Learning transferable visual models from natural language supervision". In: *International conference on machine learning*. PMLR. 2021, pp. 8748–8763.

[264] Alec Radford et al. "Learning transferable visual models from natural language supervision". In: *International conference on machine learning*. PMLR. 2021, pp. 8748–8763.

[265] Ilija Radosavovic et al. "Robot Learning with Sensorimotor Pre-training". In: *Conference on Robot Learning* (2023).

[266] Rafael Rafailov et al. "Offline Reinforcement Learning from Images with Latent Space Models". In: *Learning for Decision Making and Control (L4DC)* (2021).

[267] Colin Raffel et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *Journal of Machine Learning Research* 21.140 (2020), pp. 1–67. URL: http://jmlr.org/papers/v21/20-074.html.

[268] Md Masudur Rahman and Yexiang Xue. "Robust policy optimization in deep reinforcement learning". In: *arXiv preprint arXiv:2212.07536* (2022).

[269] J.B. Rawlings, D.Q. Mayne, and M. Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017. ISBN: 9780975937730. URL: https://books.google.ch/books?id=MrJctAEACAAJ.

[270] Scott Reed, Konrad Żołna, Emilio Parisotto, et al. "A Generalist Agent". In: *Transactions on Machine Learning Research (TMLR)* (2022).

[271] Scott Reed et al. "A Generalist Agent". In: *Transactions on Machine Learning Research* (2022).

[272] Robin Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models*. Apr. 2022. DOI: 10.48550/arXiv.2112.10752. arXiv: 2112.10752 [cs]. (Visited on 12/10/2023).

[273] Robin Rombach et al. "High-resolution image synthesis with latent diffusion models". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 10684–10695.

[274] Erick Rosete-Beas et al. "Latent Plans for Task Agnostic Offline Reinforcement Learning". In: *Proceedings of the 6th Conference on Robot Learning (CoRL)*. 2022.

[275] Chitwan Saharia et al. "Photorealistic text-to-image diffusion models with deep language understanding". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 36479–36494.

[276] Tim Salimans and Jonathan Ho. "Progressive distillation for fast sampling of diffusion models". In: *arXiv preprint arXiv:2202.00512* (2022).

[277] Tim Salzmann et al. "Real-time neural mpc: Deep learning model predictive control for quadrotors and agile robotic platforms". In: *IEEE Robotics and Automation Letters* 8.4 (2023), pp. 2397–2404.

[278] V Sanh. "DistilBERT, A Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter". In: *arXiv preprint arXiv:1910.01108* (2019).

[279] Saumya Saxena, Mohit Sharma, and Oliver Kroemer. "Multi-Resolution Sensing for Real-Time Control with Vision-Language Models". In: *7th Annual Conference on Robot Learning*. 2023. URL: https://openreview.net/forum?id=WuBv9-IGDUA.

[280] Erik Schuitema et al. "Control delay in reinforcement learning for real-time dynamic systems: A memoryless approach". In: *2010 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2010, pp. 3226–3231.

[281] Kohei Sendai et al. "Leave No Observation Behind: Real-time Correction for VLA Action Chunks". In: *arXiv preprint arXiv:2509.23224* (2025).

[282] Younggyo Seo et al. "Reinforcement learning with action-free pre-training from videos". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 19561–19579.

[283] Nur Muhammad Mahi Shafiullah et al. *On Bringing Robots Home*. 2023. arXiv: 2311.16098 [cs.RO].

[284] Nur Muhammad Mahi Shafiullah et al. "On bringing robots home". In: *arXiv preprint arXiv:2311.16098* (2023).

[285] Dhruv Shah et al. "Gnm: A general navigation model to drive any robot". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 7226–7233.

[286] Dhruv Shah et al. "ViNT: A Foundation Model for Visual Navigation". In: *7th Annual Conference on Robot Learning*. 2023. URL: https://arxiv.org/abs/2306.14846.

[287] Rutav Shah, Roberto Martín-Martín, and Yuke Zhu. "MUTEX: Learning Unified Policies from Multimodal Task Specifications". In: *7th Annual Conference on Robot Learning*. 2023. URL: https://openreview.net/forum?id=PwqiqaaEzJ.

[288] Kenneth Shaw, Shikhar Bahl, and Deepak Pathak. "Videodex: Learning dexterity from internet videos". In: *Conference on Robot Learning*. PMLR. 2023, pp. 654–665.

[289] Noam Shazeer. "Fast transformer decoding: One write-head is all you need". In: *arXiv preprint arXiv:1911.02150* (2019).

[290] Noam Shazeer et al. "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer". In: *arXiv preprint arXiv:1701.06538* (2017).

[291] Hao Shi et al. "Memoryvla: Perceptual-cognitive memory in vision-language-action models for robotic manipulation". In: *arXiv preprint arXiv:2508.19236* (2025).

[292] Liangzhi Shi et al. "Learning adaptive dexterous grasping from single demonstrations". In: *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2025, pp. 9456–9463.

[293] Lucy Xiaoyang Shi et al. "Hi robot: Open-ended instruction following with hierarchical vision-language-action models". In: *arXiv preprint arXiv:2502.19417* (2025).

[294] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. "Cliport: What and where pathways for robotic manipulation". In: *Conference on Robot Learning*. PMLR. 2022, pp. 894–906.

[295] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. "Perceiver-actor: A multi-task transformer for robotic manipulation". In: *Conference on Robot Learning*. PMLR. 2023, pp. 785–799.

[296] Mustafa Shukor et al. "Smolvla: A vision-language-action model for affordable and efficient robotics". In: *arXiv preprint arXiv:2506.01844* (2025).

[297] Ishika Singh et al. "Progprompt: Generating situated robot task plans using large language models". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 11523–11530.

[298] Jascha Sohl-Dickstein et al. "Deep unsupervised learning using nonequilibrium thermodynamics". In: *International conference on machine learning*. PMLR. 2015, pp. 2256–2265.

[299] Jiaming Song, Chenlin Meng, and Stefano Ermon. "Denoising diffusion implicit models". In: *arXiv preprint arXiv:2010.02502* (2020).

[300] Jiaming Song et al. "Pseudoinverse-guided diffusion models for inverse problems". In: *International Conference on Learning Representations*. 2023.

[301] Yang Song et al. "Consistency models". In: (2023).

[302] Ajay Sridhar et al. "Memer: Scaling up memory for robot control via experience retrieval". In: *arXiv preprint arXiv:2510.20328* (2025).

[303] Ajay Sridhar et al. "NoMaD: Goal Masked Diffusion Policies for Navigation and Exploration". In: *arXiv preprint arXiv:2310.07896* (2023).

[304] Andreas Steiner et al. "How to train your vit? data, augmentation, and regularization in vision transformers". In: *arXiv preprint arXiv:2106.10270* (2021).

[305] Austin Stone et al. "Open-World Object Manipulation using Pre-Trained Vision-Language Models". In: *7th Annual Conference on Robot Learning*. 2023.

[306] Austin Stone et al. "Open-world object manipulation using pre-trained vision-language models". In: *arXiv preprint arXiv:2303.00905* (2023).

[307] Pei Sun et al. "Scalability in perception for autonomous driving: Waymo open dataset". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2446–2454.

[308] Jie Tan et al. "Sim-to-real: Learning agile locomotion for quadruped robots". In: *arXiv preprint arXiv:1804.10332* (2018).

[309] Jiaming Tang et al. "VLASH: Real-Time VLAs via Future-State-Aware Asynchronous Inference". In: *arXiv preprint arXiv:2512.01031* (2025).

[310] Gemini Team et al. "Gemini: a family of highly capable multimodal models". In: *arXiv preprint arXiv:2312.11805* (2023).

[311] Gemini Robotics Team et al. "Gemini Robotics: Bringing AI into the Physical World". In: *arXiv preprint arXiv:2503.20020* (2025).

[312] Gemini Robotics Team et al. "Gemini robotics: Bringing ai into the physical world". In: *arXiv preprint arXiv:2503.20020* (2025).

[313] Gemma Team et al. *Gemma 3 Technical Report*. 2025. arXiv: 2503.19786 [cs.CL].

[314] Gemma Team et al. "Gemma: Open models based on gemini research and technology". In: *arXiv preprint arXiv:2403.08295* (2024).

[315] Octo Model Team et al. "Octo: An open-source generalist robot policy". In: *arXiv preprint arXiv:2405.12213* (2024).

[316] Physical Intelligence Team. "$\pi_{0.6}$ Model Card". In: (2025). URL: https://website.pi-asset.com/pi06star/PI06_model_card.pdf.

[317] Physical Intelligence Team. "$\pi^{\star}_{0.6}$: a VLA That Learns From Experience". In: *arXiv preprint arXiv:2511.14759* (2025).

[318] Physical Intelligence team. "$\pi_{0.6}$ Model Card". In: (2025).

[319] TRI LBM Team et al. *A Careful Examination of Large Behavior Models for Multitask Dexterous Manipulation*. 2025. arXiv: 2507.05331 [cs.RO]. URL: https://arxiv.org/abs/2507.05331.

[320] Waywe Research Team et al. *LINGO-2: Driving with Natural Language*. 2024.

[321] Ilya O Tolstikhin et al. "Mlp-mixer: An all-mlp architecture for vision". In: *Advances in neural information processing systems* 34 (2021), pp. 24261–24272.

[322] Marcel Torne et al. "MEM: Multi-Scale Embodied Memory for Vision Language Action Models". In: *arXiv preprint arXiv:2603.03596* (2026).

[323] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. Feb. 2023. DOI: 10.48550/arXiv.2302.13971. arXiv: 2302.13971 [cs]. (Visited on 12/10/2023).

[324] Samuel Triest et al. "Tartandrive: A large-scale dataset for learning off-road dynamics models". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 2546–2552.

[325] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[326] Mel Vecerik et al. "Robotap: Tracking arbitrary points for few-shot visual imitation". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 5397–5403.

[327] Ruben Villegas et al. "High fidelity video prediction with large stochastic recurrent neural networks". In: *Advances in Neural Information Processing Systems* 32 (2019).

[328] Homer Walke et al. *BridgeData V2: A Dataset for Robot Learning at Scale*. 2023. arXiv: 2308.12952 [cs.RO].

[329] Homer Walke et al. "BridgeData V2: A Dataset for Robot Learning at Scale". In: *Conference on Robot Learning (CoRL)*. 2023.

[330] Thomas J Walsh et al. "Planning and learning in environments with delayed feedback". In: *Machine Learning: ECML 2007: 18th European Conference on Machine Learning, Warsaw, Poland, September 17-21, 2007. Proceedings 18*. Springer. 2007, pp. 442–453.

[331] Lirui Wang et al. "Scaling Proprioceptive-Visual Learning with Heterogeneous Pre-trained Transformers". In: *Neurips*. 2024.

[332] Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. "Diffusion policies as an expressive policy class for offline reinforcement learning". In: *arXiv preprint arXiv:2208.06193* (2022).

[333] Wayve. *Lingo: Natural Language for Autonomous Driving*. 2023. URL: https://wayve.ai/thinking/lingo-natural-language-autonomous-driving/.

[334] Jason Wei et al. "Browsecomp: A simple yet challenging benchmark for browsing agents". In: *arXiv preprint arXiv:2504.12516* (2025).

[335] Jason Wei et al. "Emergent abilities of large language models". In: *arXiv preprint arXiv:2206.07682* (2022).

[336] Jason Wei et al. "Finetuned language models are zero-shot learners". In: *arXiv preprint arXiv:2109.01652* (2021).

[337] Chuan Wen et al. "Any-point trajectory modeling for policy learning". In: *arXiv preprint arXiv:2401.00025* (2023).

[338] Junjie Wen et al. "TinyVLA: Towards Fast, Data-Efficient Vision-Language-Action Models for Robotic Manipulation". In: *arXiv preprint arXiv:2409.12514* (2024).

[339] Hongtao Wu et al. "Unleashing Large-Scale Video Generative Pre-training for Visual Robot Manipulation". In: *International Conference on Learning Representations (ICLR)* (2024).

[340] Philipp Wu et al. "Daydreamer: World models for physical robot learning". In: *Conference on Robot Learning*. PMLR. 2023, pp. 2226–2240.

[341] Philipp Wu et al. "GELLO: A General, Low-Cost, and Intuitive Teleoperation Framework for Robot Manipulators". In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2024, pp. 12156–12163. DOI: 10.1109/IROS58592.2024.10801581.

[342] Philipp Wu et al. "Masked trajectory models for prediction, representation, and control". In: *International Conference on Machine Learning* (2023).

[343] Ted Xiao et al. "Thinking while moving: Deep reinforcement learning with concurrent control". In: *arXiv preprint arXiv:2004.06089* (2020).

[344] Tete Xiao et al. "Masked visual pre-training for motor control". In: *arXiv preprint arXiv:2203.06173* (2022).

[345] Wenli Xiao et al. "Self-improving vision-language-action models with data generation via residual rl". In: *arXiv preprint arXiv:2511.00091* (2025).

[346] Bin Xu et al. *Real-time reinforcement learning optimized energy management for a 48V mild hybrid electric vehicle*. Tech. rep. SAE Technical Paper, 2019.

[347] Charles Xu et al. "Rldg: Robotic generalist policy distillation via reinforcement learning". In: *arXiv preprint arXiv:2412.09858* (2024).

[348] Ge Yan, Kris Wu, and Xiaolong Wang. "ucsd kitchens Dataset". In: (Aug. 2023).

[349] Jonathan Yang, Chelsea Finn, and Dorsa Sadigh. "Data Analogies Enable Efficient Cross-Embodiment Transfer". In: *arXiv preprint arXiv:2603.06450* (2026).

[350] Jonathan Yang et al. "Pushing the Limits of Cross-Embodiment Learning for Manipulation and Navigation". In: *arXiv preprint arXiv:2402.19432* (2024).

[351] Jonathan Heewon Yang, Dorsa Sadigh, and Chelsea Finn. "Polybot: Training One Policy Across Robots While Embracing Variability". In: *7th Annual Conference on Robot Learning*. 2023. URL: https://openreview.net/forum?id=HEIRj51lcS.

[352] Ruihan Yang et al. *EgoVLA: Learning Vision-Language-Action Models from Egocentric Human Videos*. 2025. arXiv: 2507.12440 [cs.RO]. URL: https://arxiv.org/abs/2507.12440.

[353] Yinfei Yang et al. *Multilingual Universal Sentence Encoder for Semantic Retrieval*. July 2019. DOI: 10.48550/arXiv.1907.04307. arXiv: 1907.04307 [cs]. (Visited on 02/03/2023).

[354] Seonghyeon Ye et al. "Latent action pretraining from videos". In: *arXiv preprint arXiv:2410.11758* (2024).

[355] Seonghyeon Ye et al. *World Action Models are Zero-shot Policies*. 2026. arXiv: 2602.15922 [cs.RO]. URL: https://arxiv.org/abs/2602.15922.

[356] Seonghyeon Ye et al. "World action models are zero-shot policies". In: *arXiv preprint arXiv:2602.15922* (2026).

[357] Sarah Young et al. "Visual imitation made easy". In: *Conference on Robot learning*. PMLR. 2021, pp. 1992–2005.

[358] Fisher Yu et al. "Bdd100k: A diverse driving dataset for heterogeneous multitask learning". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2636–2645.

[359] Kuan-Ting Yu et al. "More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing". In: *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2016, pp. 30–37.

[360] Sihyun Yu et al. "Video probabilistic diffusion models in projected latent space". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 18456–18466.

[361] Tianhe Yu et al. "Combo: Conservative offline model-based policy optimization". In: *arXiv preprint arXiv:2102.08363* (2021).

[362] Tianhe Yu et al. "Scaling robot learning with semantically imagined experience". In: *arXiv preprint arXiv:2302.11550* (2023).

[363] Tianhe Yu et al. "Scaling robot learning with semantically imagined experience". In: *arXiv preprint arXiv:2302.11550* (2023).

[364] Tianyuan Yuan et al. "Fast-WAM: Do World Action Models Need Test-time Future Imagination?" In: *arXiv preprint arXiv:2603.16666* (2026).

[365] Michał Zawalski et al. "Robotic Control via Embodied Chain-of-Thought Reasoning". In: *arXiv preprint arXiv:2407.08693* (2024).

[366] Lihan Zha et al. "LAP: Language-Action Pre-Training Enables Zero-shot Cross-Embodiment Transfer". In: *arXiv preprint arXiv:2602.10556* (2026).

[367] Xiaohua Zhai et al. "Scaling vision transformers". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 12104–12113.

[368] Chubin Zhang et al. *CLAP: Contrastive Latent Action Pretraining for Learning Vision-Language-Action Models from Human Videos*. 2026. arXiv: 2601.04061 [cs.RO]. URL: https://arxiv.org/abs/2601.04061.

[369] Jintao Zhang et al. "Sageattention: Accurate 8-bit attention for plug-and-play inference acceleration". In: *arXiv preprint arXiv:2410.02367* (2024).

[370] Zhuoyang Zhang et al. "ForeAct: Steering Your VLA with Efficient Visual Foresight Planning". In: *arXiv preprint arXiv:2602.12322* (2026).

[371] Zongzheng Zhang et al. "Ta-vla: Elucidating the design space of torque-aware vision-language-action models". In: *arXiv preprint arXiv:2509.07962* (2025).

[372] Qingqing Zhao et al. "CoT-VLA: Visual Chain-of-Thought Reasoning for Vision-Language-Action Models". In: *Computer Vision and Pattern Recognition (CVPR)* (2025).

[373] Tony Z Zhao et al. "Aloha unleashed: A simple recipe for robot dexterity". In: *arXiv preprint arXiv:2410.13126* (2024).

[374] Tony Z Zhao et al. "Aloha unleashed: A simple recipe for robot dexterity". In: *arXiv preprint arXiv:2410.13126* (2024).

[375] Tony Z Zhao et al. "Learning fine-grained bimanual manipulation with low-cost hardware". In: *arXiv preprint arXiv:2304.13705* (2023).

[376] Haoyu Zhen et al. "3D-VLA: 3D Vision-Language-Action Generative World Model". In: *arXiv preprint arXiv:2403.09631* (2024).

[377] Jinliang Zheng et al. "X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model". In: *arXiv preprint arXiv:2510.10274* (2025).

[378] Ruijie Zheng et al. "TraceVLA: Visual Trace Prompting Enhances Spatial-Temporal Awareness for Generalist Robotic Policies". In: *arXiv preprint arXiv:2412.10345* (2024).

[379] Chunting Zhou et al. "Transfusion: Predict the next token and diffuse images with one multi-modal model". In: *arXiv preprint arXiv:2408.11039* (2024).

[380] Gaoyue Zhou et al. *Train Offline, Test Online: A Real Robot Learning Benchmark*. 2023. arXiv: 2306.00942 [cs.RO].

[381] Yuxiang Zhou, Yusuf Aytar, and Konstantinos Bousmalis. "Manipulator-independent representations for visual imitation". In: *arXiv preprint arXiv:2103.09016* (2021).

[382] Zhongyi Zhou et al. "ChatVLA: Unified Multimodal Understanding and Robot Control with Vision-Language-Action Model". In: *arXiv preprint arXiv:2502.14420* (2025).

[383] Minjie Zhu et al. "Scaling Diffusion Policy in Transformer to 1 Billion Parameters for Robotic Manipulation". In: *arXiv preprint arXiv:2409.14411* (2024).

[384] Xinghao Zhu et al. "Fanuc Manipulation: A Dataset for Learning-based Manipulation with FANUC Mate 200iD Robot". In: (2023).

[385] Yifeng Zhu, Peter Stone, and Yuke Zhu. "Bottom-Up Skill Discovery From Unsegmented Demonstrations for Long-Horizon Robot Manipulation". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 4126–4133.

[386] Yifeng Zhu et al. *VIOLA: Imitation Learning for Vision-Based Manipulation with Object Proposal Priors*. 2023. arXiv: 2210.11339 [cs.RO].

[387] Brianna Zitkovich et al. "Rt-2: Vision-language-action models transfer web knowledge to robotic control". In: *7th Annual Conference on Robot Learning*. 2023.

Appendix A

Appendicies for Chapter 2

A.1 Contributions

Dibya Ghosh: led model development, proposed and implemented large parts of the final model design, babysitted the training runs and touched all parts of the codebase, helped with model evaluations and tech report writing.

Homer Walke: led model evaluations, designed the main Bridge evaluation benchmark, contributed to the initial model implementation and ran many of the evals for this tech report.

Karl Pertsch: managed the overall project, led Open-X data integration and curation, led writing of this tech report, contributed to model development and implementation and ran model evaluations for the tech report.

Kevin Black: led data loading and training infrastructure, managed TPU pod training, created the project website, contributed to model development and implementation, and helped with robot evaluations and tech report writing.

Oier Mees: ran countless ablations for model development, contributed to model implementation, helped with evals and writing of the tech report.

Sudeep Dasari: contributed the model evaluations at CMU, experimented with pretrained encoders.

Joey Hejna: contributed the model evaluations at Stanford.

Tobias Kreiman: contributed model evaluations in simulated environments.

Ria Doshi: contributed to explorations into cross-morphology training.

Charles Xu: contributed the model evaluations for Berkeley Peg Insert.

Jianlan Luo: contributed the model evaluations for Berkeley Peg Insert.

You Liang Tan: helped diagnose and resolve bottlenecks in data loading.

Lawrence Yunliang Chen: helped set up and run UR5 finetuning experiments at Berkeley AutoLab.

Pannag Sanketi, Quan Vuong, Ted Xiao: contributed model evaluations on the Google Robot.

Dorsa Sadigh, Chelsea Finn, Sergey Levine: provided guidance throughout the project and feedback on the writing of this tech report.

A.2 Octo Code Example

Loading a pretrained Octo model and performing inference requires little code:

Listing A.1: Example Python code to perform inference with a pretrained Octo model.

```
import jax
from octo.model.octo_model import OctoModel

model = OctoModel.load_pretrained("hf://rail-berkeley/octo-base")
print(model.get_pretty_spec()) # Print out the input-output spec
observation = {"image_primary": img}
task = model.create_tasks(texts=["pick␣up␣the␣fork"])
action = model.sample_actions(
    observation, task, rng=jax.random.PRNGKey(0))
```

A.3 Data mixture

We list the detailed training mixture used for training the Octo models in Table A.1. The sampling weights are mostly determined by the relative size of the datasets with a few manual adjustments (see Section 2.3).

A.4 Training Hyperparameters

We mostly follow documented practices for training vision transformers [367]. We use the AdamW optimizer [202] with an inverse square root decay learning rate schedule [367] and linear learning rate warm-up. We list hyperparamaters used during training in Table A.2

Octo Pretraining Dataset Mixture	
Fractal [42]	17.0%
Kuka [146]	17.0%
Bridge[85, 328]	17.0%
BC-Z [136]	9.1%
Stanford Hydra Dataset [17]	6.0%
Language Table [208]	5.9%
Taco Play [274, 222]	3.6%
Furniture Bench Dataset [119]	3.3%
UTAustin Mutex [287]	3.0%
Austin Sailor Dataset [237]	2.9%
Roboturk [217]	2.8%
Toto [380]	2.4%
Austin Sirius Dataset [195]	2.3%
Berkeley Autolab UR5 [55]	1.5%
IAMLab CMU Pickup Insert [279]	1.2%
Viola [386]	1.2%
Berkeley Fanuc Manipulation [384]	1.0%
NYU Franka Play Dataset [68]	0.9%
UCSD Kitchen Dataset [348]	<0.1%
Jaco Play [70]	0.6%
Berkeley Cable Routing [206]	0.3%
Austin Buds Dataset [385]	0.3%
CMU Stretch [227]	0.2%
NYU Door Opening [246]	0.1%
DLR EDAN Shared Control [261]	0.1%

Table A.1: Octo pretraining data mixture using datasets from the Open X-Embodiment dataset [240].

and the model parameters for the different sizes in Table A.3. We apply standard image augmentations during training. Concretely, for the 3rd person camera we apply stochastic crops followed be a resize to 256×256, followed by color jitter. Finally, we normalize the input image to have pixels with float values between -1.0 and 1.0. For the wrist camera, we apply the same procedure except without the random crop and resizing to 128×128 instead.

The images are passed through a shallow convolution stack, then split into a sequence of flattened patches [75] of size 16×16. This results in 256 tokens for the 3rd person camera images and 64 tokens for the wrist camera images. For datasets containing language annotations, we use a pretrained `t5-base` (111M) transformer model [267] that produces a

Hyperparameter	Value
Learning Rate	3e-4
Warmup Steps	2000
LR Scheduler	reciprocal square-root
Weight Decay	0.1
Gradient Clip Threshold	1
Batch Size	2048

Table A.2: Hyperparameters used during training.

sequence of 16 language embedding tokens.

Model	Layers	Hidden size D	MLP size	Heads	Params
Octo-Small	12	384	1536	6	27M
Octo-Base	12	768	3072	12	93M

Table A.3: Architecture details of Octo model variants.

The diffusion action head consists of a 3-layer MLP with a hidden dimension of 256, residual connections, and layer normalization. We use the standard DDPM objective as introduced by [120] with a cosine noise schedule [238] and 20 diffusion steps.

A.5 Things that Worked and Did Not Work (Yet)

Things we found improved performance

- **Adding history during pretraining**: Models with one frame of history as context performed better in zero-shot evals than models pretrained without history. We did not observe benefits of increasing the history length further on the few tasks we evaluated on, though other tasks may benefit.
- **Using action chunking**: We found it helpful to use "action chunking" [375], i.e., to predict multiple actions into the future, for getting more coherent policy movements. In our evaluations, we did not find temporal ensembling of future actions to provide additional benefits beyond receding horizon control.
- **Decreasing patch size** Tokenizing images into patches of size 16×16 led to improved performance over patches of size 32×32, particularly for grasping and other fine-

grained tasks. This does add compute complexity (the number of tokens is 4×), so understanding how to balance compute costs and resolution remains a problem of interest.

- **Increasing shuffle buffer size**: Loading data from 25 datasets in parallel is a challenge. Specifically, we found that achieving good shuffling of frames during training was crucial — zero-shot performance with a small shuffle buffer (20k) and trajectory-level interleaving suffered significantly. We solved this issue by shuffling and interleaving frames from different trajectories *before* decoding the images, allowing us to fit a much larger shuffle buffer (up to 500k). We also subsample at most 100 randomly chosen steps from each training trajectory during data loading to avoid "over-crowding" the shuffle buffer with single, very long episodes.

Things that did not work (yet)

- **MSE Action Heads**: Replacing our diffusion decoding head with a simple L2 loss led to "hedging" policies that move very slowly and e.g., fail to rotate the gripper in WidowX evaluations.
- **Discrete Action Heads**: Discretizing actions into 256 bins per dimension and training with cross-entropy loss like in Brohan et al. [42] led to more "decisive" policies, yet they lacked precision and often missed the grasp.
- **ResNet Encoders**: did not scale as well to larger datasets in our evaluations (see Table 2.2), though they did outperform our ViT architecture when training from scratch on a small dataset (around 100 demonstrations).
- **Pretrained Encoders**: ImageNet pretrained ResNet encoders did not provide benefit on zero-shot evals, though may be confounded with ResNet architectures underperforming as mentioned above.
- **Relative Gripper Action Representation**: When aligning the gripper action representations of the different datasets, we tried (A) absolute gripper actions, i.e., actions are +1 when the gripper is open and 0 if it is closed, and (B) relative gripper actions, i.e., gripper action is +1/0 only in the timestep when the gripper opens/closes and 0.5 otherwise. We found that the latter tends to open/close the gripper less often since most of the training data represents "do not change gripper" actions, leading to a slightly higher grasp success rate. At the same time, the relative representation led to less retrying behavior after a grasp failed, which was ultimately worse. Thus, we chose the absolute gripper action representation.
- **Adding Proprioceptive Inputs**: Policies trained with propioceptive observations seemed generally worse, potentially due to a strong correlation between states and

future actions. We hypothesize this might be due to a causal confusion between the proprioceptive information and the target actions [110].

- **Finetuning Language Model**: In order to improve the visuo-lingual grounding of Octo we experimented with: i) varying sizes of the T5 encoder [267]: `small` (30M), `base` (111M), and `large` (386M) as well as ii) finetuning the last two layers of the encoder. Using the frozen `base` model resulted in the best language-conditioned policies. We did not find improvements when using larger encoders or finetuning the encoder. We hypothesize this might be due to the lack of rich, diverse, free-form language annotations in most of the datasets.

A.6 Experimental Setups

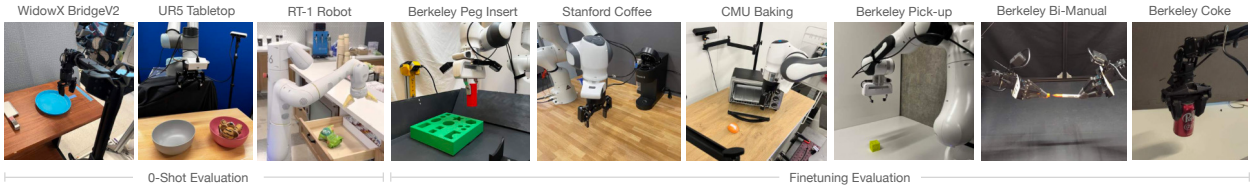


Figure A.1: **Evaluation Tasks.** Replicated from the main text for convenience. We evaluate Octo on 9 real robot setups across 4 institutions in zero-shot and finetuning scenarios.

Zero-Shot Evaluations

WidowX BridgeV2 Uses the setup of Walke et al. [328], in which a Trossen WidowX 250 6-DOF robot performs diverse manipulation tasks. The observation consists of a single third person camera stream and the action space is end-effector position deltas. We evaluated two language-conditioned tasks in which a the robot needs to "place the carrot on plate", and "put the eggplant in the pot." While these tasks are in-distribution, the policy must still generalize to novel object positions. We performed 10 trials per task and varied objects positions between trials.

UR5 Uses the setup of Chen, Adebola, and Goldberg [55], in which a UR5 robot arm performs multiple table top manipulation tasks. The observation consists of a single third person camera stream and the action space is end-effector position deltas. We evaluated twp language-conditioned tasks: picking a toy tiger from a bowl and placing it into a different bowl as well as wiping a table with a cloth. While these tasks are in-distribution, the policy must still generalize to novel object positions, distractor objects, and lighting. Since the training data was collected months ago and the robot setup was taken down and re-assembled, the policy must also generalize to other miscellaneous changes in the environment like a slightly different camera view and background. We performed 10 trials per task and varied objects positions between trials.

		Language-conditioned		Goal-conditioned		
		Put carrot on plate	Put eggplant in pot	Put bread on plate	Put spoon on glove	Average
	Octo-small (Ours)	80%	90%	70%	90%	**83%**
DATA	RT-X dataset mix [240]	80%	80%	40%	40%	60%
DATA	Single robot dataset (Bridge Data)	20%	70%	60%	20%	43%
POLICY	Discretized Action Prediction [240]	0%	20%	10%	40%	18%
POLICY	Continuous Action Prediction (MSE)	70%	30%	0%	40%	35%
ARCH	Resnet-50 + Transformer [240]	80%	60%	100%	40%	70%

Table A.4: **Model Ablations.** We achieve best performance when using the ViT architecture, diffusion action head, and wide training data mixture. All evaluations are performed on the WidowX setup. Success rates are averaged over 40 trials across two language-conditioned tasks and two goal-conditioned tasks.

RT-1 Robot Uses the setup of Brohan et al. [42], in which a proprietary robot performs multiple table top and furniture manipulation tasks. The observation consists of a single third person camera stream and the action space is end-effector position deltas. We evaluated on the task of picking up a 7up can, apple, blue chip bag, or brown chip bag, as well as the task of opening or closing drawers on a cabinet. While these tasks are in-distribution, the policy must still generalize to novel object positions. Additionally, since the robot setup was moved between buildings, the policy must also generalize to miscellaneous changes in the environment like a slightly different camera view and background. We performed 10 trials per task and varied objects positions between trials.

Model Ablations

All of our model ablations were evaluated on the WidowX setup. We present a more detailed breakdown of the success rates per task in Table A.4. We evaluated on two language-conditioned tasks (put carrot on plate and put eggplant in pot) and two goal-conditioned tasks (put bread on plate and put spoon on glove). The goal-conditioned tasks contain objects not seen in the Bridge dataset. Additionally, we analyze the generalization capabilities of Octo across several different axes, such as novel objects, novel environments and novel skills in Table A.5.

Finetuning Evaluations

CMU Baking The robot must pick up the toy bread object, place it in the toaster, and shut the toaster. This task requires generalization across initial positions (of both the toaster and object) and the shape of the target toy bread object. We use an end-effector delta action space (Cartesian position + rotation delta). Observations come from the 3rd-person front-facing Zed camera. Actions are predicted at 15 Hz, and executed on the robot using the

Generalization Type	Task	Success Rate	Average
In-distribution	Put carrot on plate Put eggplant in pot	80% 90%	85%
Novel objects	Put bread on plate Put spoon on glove	70% 90%	80%
Novel environment	Put mushroom in pot Put spoon on cloth	20% 60%	40%
Novel skill	Flip cup on its side Put block in slot	10% 0%	5%

Table A.5: **Zero-shot Generalization Analysis.** Using the WidowX setup, we analyze the generalization capabilities of Octo-small along several different axes. Octo performs the best on in-distribution tasks and generalizes well to novel objects and environments. However, Octo struggles to generalize to skills not seen in the WidowX embodiment. Success rates are averaged over 20 trials across two tasks.

R2D2 Franka controller. The finetuning dataset consists of 120 demos collected via expert VR tele-operation, and every policy was evaluated using 20 trials (4 novel test objects with 5 positions each).

Stanford Coffee The robot is tasked with picking up one of four different Keurig Coffee Pods and placing it inside of a Keurig machine. This task requires both generalization across initial positions and colors of the coffee pod, as well as precise placement in the Keurig machine. We use an end effector delta action space with an open source controller running at 10 Hz based on Polymetis [191] (found here). We use only a single 3rd-person wrist observation. Our training dataset contained 118 expert demonstrations from varied coffee pods and positions collected via VR tele-operation. We evaluated policies for 20 episodes, five episodes for each of four different color coffee pods.

Berkeley Peg Insertion The task is to insert a pre-grasped 3D-printed peg into a matching slot on a 3D-printed board inside the bin. The matching tolerance between the peg and the hole is 1.5mm; which makes it a contact-rich precise part-mating task. The robot must learn an appropriate policy to “search" for the matching opening through contact, which necessitates the use of force/torque measurements. The observation space of the policy consists of a single side-view camera image, the end-effector twist, and the end-effector force/torque reading. The policy sends action commands as the robot's end-effector twists at 5 HZ, tracked at 1000 HZ by a low-level impedance controller. Our finetuning dataset is composed of 100 human demonstrations from the FMB dataset [205]. We evaluated trained

policies for 20 trials with randomized board positions.

Berkeley Pick Up We use the setup of Radosavovic et al. [265]. The robot needs to pick up a block from a table top surface after being trained on a dataset of 100 pickups of various objects. The robot uses joint position control with an underlying Polymetis control stack [191] (here). It is conditioned on a wrist camera input image, as well as the proprioceptive readings of the robot. We evaluted on the task of picking up the yellow cube and used 20 trials.

Berkeley Coke A Trossen Robotics ViperX robot is tasked with picking up a coke can from a table. The ViperX is completely unseen in the training mixture, thus this task tests the model's capabilities to generalize to a new robot embodiment. The task also requires generalization across different positions of the coke can with the ViperX robot using end-effector delta control at 5Hz. The policy uses image observations from a third person and a wrist camera. Our training dataset contained 115 expert demonstrations collected via VR tele-operation from varied positions.

Berkeley Bimanual The task requires an ALOHA bimanual robot [375], consisting of two ViperX robot arms, to pick up a sharpie marker with the right hand from the workbench and remove its cap with the left hand. Since Octo was only pre-trained with single-arm robot data, we re-initialize a new action head that maps to the 14-dimensional action space of ALOHA (2x 6 joint positions + 2x gripper position). A successful policy needs to use image observations from the left and right wrist cameras to predict precise manipulation behaviors. Best performance for both our methods and baselines was achieved by predicting an action chunk of 64 during training, and then during test-time executing 12 actions with receding horizon control before re-planning. We evaluated the uncapping task using 10 trials, where the pen was placed in a different location for each trial.

Appendix B

Appendices for Chapter 3

B.1 Contributions

The authors contributed to the following areas (listed alphabetically):

Data and operations: Noah Brown, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Liyiming Ke, Suraj Nair, Lucy Shi, and Anna Walling.

Evaluation experiments: Kevin Black, Michael Equi, Chelsea Finn, Brian Ichter, Liyiming Ke, Adrian Li-Bell, Suraj Nair, Karl Pertsch, and Lucy Shi.

Model design: Kevin Black, Brian Ichter, Sergey Levine, Karl Pertsch, Lucy Shi, and Quan Vuong.

Post-training: Michael Equi, Chelsea Finn, Liyiming Ke, Adrian Li-Bell, Suraj Nair, and Lucy Shi.

Pre-training: Kevin Black, Danny Driess, Brian Ichter, Sergey Levine, Karl Pertsch, Lucy Shi, and Quan Vuong.

Robot hardware: Noah Brown, Adnan Esmail, Chelsea Finn, Tim Jones, and Mohith Mothukuri.

Robot software: Karol Hausman, Szymon Jakubczak, Sergey Levine, James Tanner, and Haohuan Wang.

Training infrastructure: Kevin Black, Michael Equi, Sergey Levine, Adrian Li-Bell, Suraj Nair, Quan Vuong, Haohuan Wang, and Ury Zhilinsky.

Writing and illustration: Kevin Black, Chelsea Finn, Lachy Groom, Karol Hausman, Brian Ichter, Sergey Levine, and Quan Vuong.

B.2 Model Architecture Details

In this section, we provide a full description of the model architecture. We follow the PaliGemma VLM [19] design, with the following differences: (1) additional input and output projections for the robotics-specific tokens, including the state vector $\mathbf{q}_t$ and action vectors $\mathbf{A}_t = [\mathbf{a}_t, ..., \mathbf{a}_{t+H-1}]$, (2) an additional MLP for incorporating the flow matching timestep information τ, and (3) a second, smaller set of weights for the action expert.

Additional inputs and outputs. The standard PaliGemma architecture takes in a sequence of images $[\mathbf{I}_t^1, ..., \mathbf{I}_t^n]$ followed by a language prompt ℓ_t. We add an input $\mathbf{q}_t$ for the robot's proprioceptive state, which is mapped to the transformer embedding dimension using a linear projection. The final set of input tokens correspond to the noisy action chunk $\mathbf{A}_t^\tau = [\mathbf{a}_t^\tau, ..., \mathbf{a}_{t+H-1}^\tau]$, with the number of tokens equal to the action horizon ($H = 50$ for our tasks). We only use the transformer outputs corresponding to the H noisy actions, which are decoded into $\mathbf{v}_\theta(\mathbf{A}_t^\tau, \mathbf{o}_t)$ using a linear projection.

Incorporating the flow matching timestep. The noisy action chunk $\mathbf{A}_t^\tau$ is mapped to the transformer's embedding dimension using an MLP that also incorporates the flow matching timestep τ. For each noisy action $\mathbf{a}_{t'}^\tau$, the expression for the corresponding embedding that is fed into the transformer is $W_3 \cdot \text{swish}(W_2 \cdot \text{concat}(W_1 \cdot \mathbf{a}_{t'}^\tau, \phi(\tau)))$, where $\phi : \mathbb{R} \to \mathbb{R}^w$ is a sinusoidal positional encoding function [325], $W_1 \in \mathbb{R}^{w \times d}$, $W_2 \in \mathbb{R}^{w \times 2w}$, $W_3 \in \mathbb{R}^{w \times w}$, d is the action dimension, and w is the embedding dimension (or *width*) of the action expert.

Attention mask. π_0 uses a blockwise causal attention mask with 3 blocks: $[\mathbf{I}_t^1, ..., \mathbf{I}_t^n, \ell_t]$, $[\mathbf{q}_t]$, and $[\mathbf{a}_t^\tau, ..., \mathbf{a}_{t+H-1}^\tau]$. Within each block, there is full bidirectional attention, whereas the tokens in each block cannot attend to the tokens in future blocks. The first block includes the input modalities from PaliGemma's VLM pre-training, which are prevented from attending to future blocks (which include new inputs) to minimize distribution shift from said pre-training. The robot state $\mathbf{q}_t$ is its own block because it does not change with each flow matching integration step; preventing it from attending to the final block allows its corresponding keys and values to be cached during sampling. The final block corresponds to the noisy actions $\mathbf{A}_t^\tau$, which can attend to the full input sequence.

Action expert. π_0 is implemented as a single transformer with two sets of weights (also known as experts [290]), where each token is routed to one of the experts; the weights interact only through the transformer's self-attention layers. The images and language prompt, $[\mathbf{I}_t^1, ..., \mathbf{I}_t^n, \ell_t]$, are routed to the larger VLM backbone, which we initialize from PaliGemma. The inputs not seen during VLM pre-training, $[\mathbf{q}_t, \mathbf{A}_t^\tau]$, are routed to the

action expert. PaliGemma is based on the Gemma 2B [314] language model, which uses multi-query attention [289] and a configuration of {*width*=2048, *depth*=18, *mlp_dim*=16,384, *num_heads*=18, *num_kv_heads*=1, *head_dim*=256}. Since the experts interact only in the self-attention layers, *width* and *mlp_dim* do not necessarily need to match between experts. To speed up inference (which requires multiple forward passes of the action expert), we downsize the action expert to {*width*=1024, *mlp_dim*=4096}, resulting in a parameter count of ∼300M.

Sampling the flow matching timestep. The original flow matching papers [192, 197] sample the flow matching timestep from a uniform distribution: $\tau \sim \mathcal{U}(0, 1)$. Esser et al. [88] instead propose sampling from a logit-normal distribution that emphasizes the middle timesteps; the authors posit that at high timesteps (low noise levels), the model needs only to learn the identity function, and at low timesteps (high noise levels), the model needs only to learn the mean of the data distribution. However, we hypothesize that the task of action prediction is subtly different from high-resolution image synthesis — while it may be relatively easy to predict the mean image conditioned on a text label, predicting the mean action conditioned on a robot observation (i.e., learning $\mathbb{E}[\mathbf{A}_t|\mathbf{o}_t]$) is a much harder problem; this is because the observation $\mathbf{o}_t$ is very *informative* in that it should constrain the distribution of possible actions much more than a text label constrains the distribution of possible images. As a result, we designed a timestep sampling distribution that emphasizes low timesteps (high noise levels); additionally, timesteps above a given threshold s are not sampled at all, since they are not needed so long as the integration step δ is greater than $1 - s$. The distribution is given by $p(\tau) = \text{Beta}(\frac{s-\tau}{s}; 1.5, 1)$ and is visualized in Figure B.1. We use $s = 0.999$ in our experiments, which allows for $\delta > \frac{1}{1000}$, or up to 1,000 integration steps.

B.3 Non-VLM Baseline Architecture

Our baseline architecture π_0-small is *not* based on a VLM backbone. Hence, we use it to evaluate the benefits of VLM-pre-training. We design it to be sufficiently expressive to fit our large dataset while still providing good performance when trained from scratch. This model has about 470M parameters, and differs from our main model in the following ways: (1) We use DistilBERT [278] to encode the language tokens of the language command ℓ_t, since this model does not use a language model backbone; (2) The action expert cross-attends to the outputs of the observation encoder, akin to a traditional encoder-decoder transformer [325], rather than our main model which is more like a decoder-only mixture of experts [290]; (3) The images are encoded with a smaller pre-trained ViT encoder (specifically, the R26-S-32 ResNet-ViT hybrid from Steiner et al. [304]); (4) The ViT image encoders do not share weights; (5) The transformer backbone that encodes the observations (which comes after the ViT image encoders) is not pre-trained on Internet data; (6) The action expert uses the DiT architecture [250] rather than the Gemma architecture, and hence incorporates the flow-

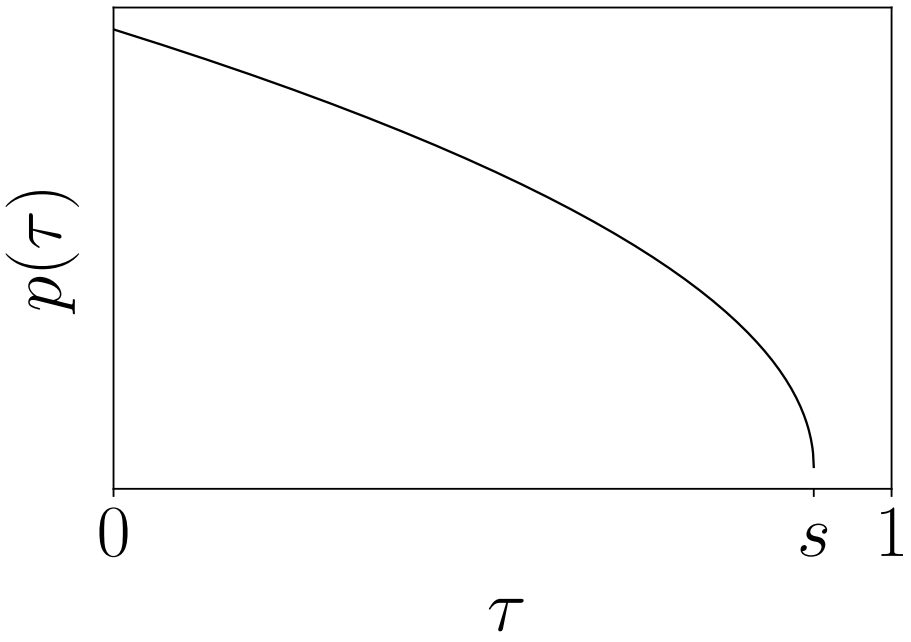


Figure B.1: **Flow matching timestep sampling distribution**. We sample τ from a shifted beta distribution that emphasizes lower timesteps (corresponding to noisier actions), and does not sample timesteps at all above a cutoff value s. We use $s = 0.999$ in our experiments.

matching timestep τ using AdaLN-Zero layers. Besides this, the models are broadly similar: both use pre-trained ViT image encoders, both use separate weights for the observation encoder and the action expert, both take in the same observation format, and both perform 10 steps of flow matching to predict the action chunk.

B.4 Inference

Recall that our model takes an observation $\mathbf{o}_t = [\mathbf{I}_t^1, ..., \mathbf{I}_t^n, \ell_t, \mathbf{q}_t]$ and the noisy actions $\mathbf{A}_t^\tau$ and outputs the vector field that needs to be integrated to obtain the next flow matching step, $\mathbf{v}_t^\tau$. Each time we predict a new action chunk $\mathbf{A}_t$, we must encode each of the images $\mathbf{I}_t^1, ..., \mathbf{I}_t^n$, run a forward pass on the tokens corresponding to $\mathbf{o}_t$, and then run 10 steps of flow matching, where each step requires running a forward pass on the tokens corresponding to $\mathbf{A}_t^\tau$ (the keys and values corresponding to $\mathbf{o}_t$ are cached). Table B.1 summarizes the computation time for this operation with 3 camera images. The operations were timed on an NVIDIA GeForce RTX 4090 consumer-grade GPU. For the mobile robot, inference was done off-board over a Wi-Fi connection, adding a small amount of network latency. Further optimizations, quantization, and other improvements might further reduce inference times.

Since the model generates an entire H-step action chunk at once, we can execute up to H actions before we need to run inference again. However, we may run inference more often than that, as well as combine actions from different inference calls using various aggregation strategies. We tried temporal ensembling [375] early on and found that it hurt policy performance, so we opted not to aggregate actions and instead execute action chunks open-loop. For the 20Hz UR5e and Franka robots, we run inference every 0.8 seconds (after executing 16 actions), and for all other robots, which run at 50Hz, we run inference every 0.5 seconds (after executing 25 actions).

model part	inference time
image encoders	14 ms
observation forward pass	32 ms
x10 action forward pass (flow)	27 ms
network latency (if off-board)	13 ms
total on-board inference	73 ms
total off-board inference	86 ms

Table B.1: Inference time of our model on an NVIDIA GeForce RTX 4090 GPU.

B.5 Evaluation Details

For each task, we design a score rubric that measures progress on the task, and use this for our quantitative results. We describe this rubric for each task below:

A. Evaluating the base model

Shirt folding: Shirt folding is recorded as either success or failure. We begin each shirt folding eval by laying the shirt flat on the table. Success is defined as having folded in the sleeves and performed one half-fold along the length of the shirt. Our eval includes 4 small t-shirts and 1 medium t-shirt. We run 2 evals for each item for a maximum of 15000 steps or approximately 5 minutes each.

Bussing easy: This task is scored out of 7, where there are 7 different objects on the table, and 1 point is given for each correctly sorted object.

Bussing hard: This task is scored out of 12, where there are 12 different objects on the table, and 1 point is given for each correctly sorted object. This version of the task includes particularly challenging settings, like a chopstick on top of a piece of trash.

Grocery bagging: This task is scored out of 7. For each 7 grocery items, a point is given for putting it in the bag.

Toast out of toaster: This task is scored out of 4. For each piece of toast, 1 point is given for picking it from the toaster and another for putting it on the plate.

B. Language instruction following. The policy is scored on successfully repositioning each object and whether it follows instructions.

Bussing: The robot has to follow the command to pick up the correct object and place each of them into the correct receptacle. The robot receives 12 objects in total and around 30 instructions in one episode.

Table setting: The robot arranges all dishes, utensils, and napkins and makes adjustments according to language specification. The robot receives 7 objects in total and around 20 instructions in one episode.

Grocery bagging: The robot picks up the correct item (among bag of coffee beans, bag of barley, bag of marshmallow, cat food, spaghetti, bag of seaweed, bag of almonds), and bags them into a paper bag. The robot receives 7 objects in total and around 14 instructions in one episode.

C. Learning new dexterous tasks

Stack bowls: This task is scored out of 3. One point for each of two bowls stacked in larger bowls, and one for the neatness of the final product.

Towel folding: This task is scored out of 3. One point for the first half-fold of the towel, one point for the second half-fold of the towel, and one point for neatness of the final product.

Tupperware in microwave: This task is scored out of 4. One point for opening the microwave, one point for picking up the Tupperware, one point for putting the Tupperware in the microwave, and one point for closing the microwave.

Paper towel replacement: This task is scored out of 4. One point is given for grasping the old roll, and another point is given for removing it. Then, one point is given for grasping the new paper towel roll, and the final point is given for placing it on the dispenser.

Items in drawer: This task is scored out of 5. One point for opening the drawer, one point for each of 3 items picked and placed into the drawer, and one point for closing the drawer.

D. Mastering complex multi-stage tasks

Laundry folding: This task is scored out of 4. Our evaluation includes five items, three shirts of size M, L, and XL and two shorts of size 28 and 36. We perform two trials for each item, and the items left to be evaluated start randomly crumpled in a laundry bin (while previously evaluated items start in a folded stack). One point is given for picking an item out of the bin and putting it on the table. Another point is given for flattening the shirt or shorts. A third point is granted for folding the shirt or shorts. A final point is given for either placing the item in the corner of the table (if it is the first item evaluated), or stacking it onto an existing stack of folded clothes. We run each eval for a maximum of 15000 steps or approximately 5 minutes.

Mobile laundry: This evaluation follows the same protocol as laundry folding. The three shirts are sized M, M, and XL, and the shorts are sized 32 and 31 W.

Table bussing: This task is scored out of 12, where there are 12 different objects on the table, and 1 point is given for each correctly sorted object. This version of the task includes particularly challenging settings, like a chopstick on top of a piece of trash.

Box building: This task is scored out of 5. One point is given for successfully picking up the box to begin the task. One point is given for folding the box in half, so the flaps can be closed. One point is given for closing the right flap. One point is given for closing the left flap. The final point is given for neatly centering the final product.

Packing eggs: This task is scored out of 7. One point for each egg placed in the correct slot in the carton, and one point for closing the lid.

Packing food: This task is scored out of 5. One point for picking up the plate of food, one point for each of 3 food items placed in the to-go box, and one point for closing the to-go box.

Dryer unloading: This task involves having the robot approach a dyer with a laundry basket and unload the clothes into the basket. We score this eval out of five, where one point is given for properly approaching the dryer. Another for placing the laundry basket on the stool. A third for opening the dryer. A fourth for putting all the clothes in the basket and a fifth point for closing the dryer. We eval with 3 shirts and 2 shorts that start in a random configuration inside the dryer.

Appendix C

Appendices for Chapter 4

C.1 Broader Impacts

The goal of our work is to improve the speed and performance of learned policies for control tasks, and our experiments primarily deal with household robots. This technology has great potential to improve lives, e.g., by automating dangerous and difficult jobs, or assisting the disabled and elderly. Like any technology, it also has the potential for harm—e.g., in military applications, or by displacing physical labor.

C.2 The Necessity of Guidance Weight Clipping (β)

In Section 4.4, we describe how we adapt the inpainting algorithm from Pokle et al. [257] and Song et al. [300] to our setting. One modification we make is to add a clipping value, β, which limits weight applied to the guidance term (Eq. 4.2), and is necessary to make the weight finite at $\tau = 0$[1]. While image inpainting typically uses a high number of denoising steps (e.g., $n = 100$ in [257]), control problems often use very few steps (e.g., $n = 5$ in our experiments). In this case, we found that high guidance weights led to diverging action chunks, as shown in Figure C.1, bottom left. Based on a simulated ablation (Figure C.1, top right), we set β to a conservative value of 5.

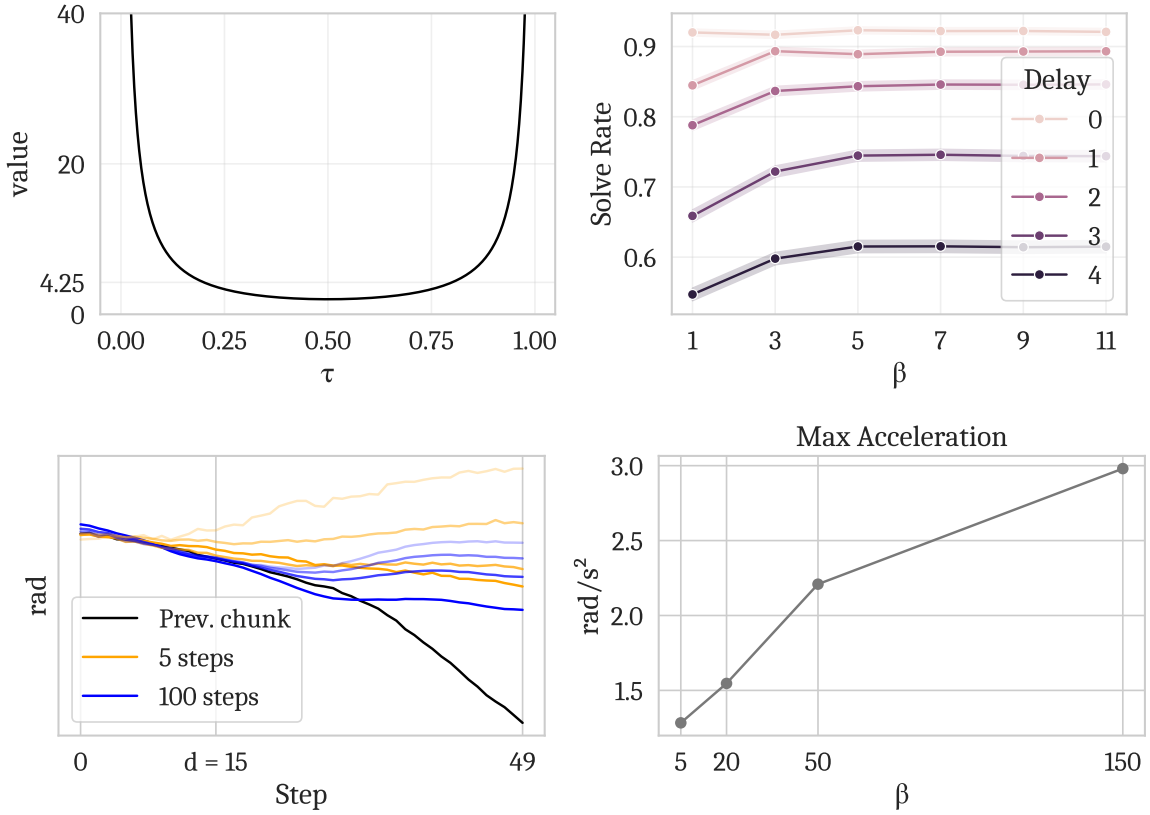


Figure C.1: **Top left:** The graph of the value $\frac{1-\tau}{\tau \cdot r_\tau^2}$ from Eq. 4.2, which we clip at β. At $\tau = 0$, clipping is needed to make the value finite. With 5 denoising steps, if $\beta \geq 4.25$, the clipping only determines the guidance weight for the first step ($\tau = 0$). **Top right:** An ablation of β in our simulated benchmark. Increasing β provides no marginal benefit beyond $\beta = 5$. **Bottom left:** Example real robot action chunks generated from the same noise with 5 denoising steps ($n = 5$) and 100 denoising steps ($n = 100$), with lower opacities corresponding to higher guidance weight clipping ($\beta = \{5, 20, 50, 150\}$). With 5 denoising steps, the generated action chunks diverge when β is too high. **Bottom right:** β vs. maximum acceleration (second discrete difference) for a batch of 325 action chunks generated with $d = 15$ and $n = 5$. Higher β leads to more jerkiness, a proxy for out-of-distribution actions.

Method	Latency
RTC (ours)	97ms
BID with $N = 16$ (no forward model)	115ms
BID with $N = 16$ (shared backbone)	169ms
BID with $N = 16$ (full)	223ms
Vanilla $\pi_{0.5}$	76ms

Table C.1: Latency measurements for various inference-time methods applied to $\pi_{0.5}$ [134]. Numbers include on-GPU neural network inference only, and are averaged over 10 inference calls after 5 warmup calls. Inference runs on an NVIDIA RTX 4090 GPU using bfloat16 precision and $n = 5$ denoising steps. BID [201] slows down inference due to sampling batches of actions, whereas RTC slows down inference due to backpropagating through each denoising step. BID (no forward contrast) refers to a version of BID without the forward contrast loss, which elides the need for a second model. BID (shared backbone) refers to a version of BID optimized specifically for the π_0 architecture, where the VLM backbone (3B parameters) is shared between the strong and weak model, so only two copies of the action expert (300M parameters) are needed. Full BID requires two copies of the entire model.

C.3 Latency Measurements

C.4 Additional Simulated Ablations

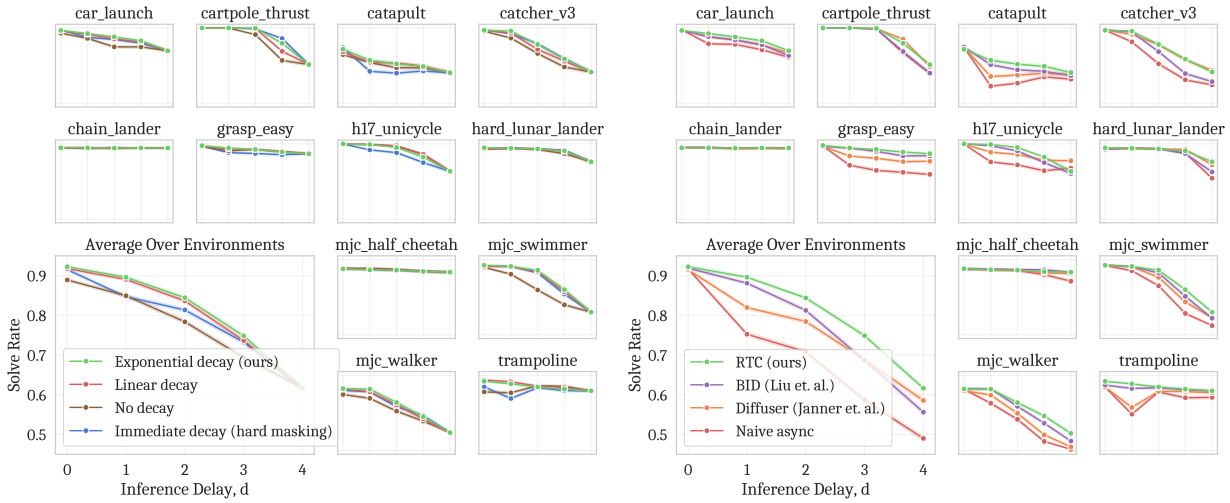


Figure C.2: **Left:** Simulated ablation over different schedules for soft masking weights (Eq. 4.5). Exponential decay performs the best overall, although linear decay is very close behind. **Right:** Comparison with the inpainting algorithm from Diffuser [140], which overwrites a portion of the action chunk with the desired actions at each denoising step. While this simpler (and cheaper) inpainting method still provides some benefit, it is outperformed by our guidance-based approach.

[1] An alternative approach to avoid the infinite weight at $\tau = 0$ is to start denoising from $\tau > 0$, used in [257], which we did not try.

Component	Time (mobile)	Time (non-mobile)
Model	96.89 ± 0.16ms	97.43 ± 0.28ms
Network	21.20 ± 3.12ms	6.89 ± 2.39ms
Image resize	11.22 ± 5.00ms	1.44 ± 0.27ms
Other	9.67 ± 3.20ms	3.00 ± 0.68ms
Total	138.98 ± 6.71ms	108.76 ± 2.34ms

Table C.2: Breakdown of **total** inference latency by component for RTC. The image resizing component happens on the CPU of the robot computer. In the mobile manipulation case, this computer is an Intel NUC portable computer with a 12th Gen Intel i7-1260P processor. In the non-mobile case, this computer is a desktop workstation with an AMD Ryzen 9 7950X processor. In both cases, the model runs on a separate workstation with an NVIDIA RTX 4090 GPU; the robot computer and the inference workstation are both connected to the same LAN via a wired Ethernet connection, and communication happens via the WebSocket protocol. Model inference uses bfloat16 precision and $n = 5$ denoising steps. Measurements are taken from 50 inference calls during a real episode rollout, and ± one standard deviation is shown.

Component	Time (no RTC)	Time (with RTC)
Image encoders (SigLIP)	18ms	18ms
LLM prefill (Gemma 2B)	44ms	44ms
Denoising step (x5)	14ms	35ms
Total	76ms	97ms

Table C.3: Breakdown of **model** inference latency by component for vanilla $\pi_{0.5}$ and RTC. Measurements are taken from a single profiling trace for each method, run on an RTX 4090 GPU. RTC incurs a 2.5x latency increase per denoising step.

C.5 Hyperparameters

Hyperparameter	Description	Simulation	Real-world
n	Denoising steps	5	5
H	Prediction horizon	8	50
$s_{\min}$	Minimum execution horizon	-	25
β	Guidance weight clipping	5	5
b	Delay buffer size	-	10

Table C.4: Hyperparameters used for RTC (Algorithm 1). In simulation, d is held constant for each experiment, so $s_{\min}$ and b are not needed. Additional hyperparameters for the simulated experiments can be found in the code.

C.6 Code Release

The code for the simulated experiments is available at `https://github.com/Physical-Intelligence/real-time-chunking-kinetix`.

C.7 Compute Resources

All the experiments in this work use no more than 8 NVIDIA H100 GPUs (one NVIDIA DGX server) at a time. H100s are used via a cloud provider.

Simulated experiments. Training expert policies with RPO [268] with 6 seeds $\times$ 12 environments takes approximately 4 hours on 4xH100s. Generating data from those policies takes approximately 20 minutes on 6xH100s. Training imitation learning policies with flow matching for each environment takes approximately 1.5 hours on 2xH100s. Evaluating the policies for 2048 trials per environment takes approximately 5 minutes on 6xH100s.

Real-world experiments. We use policies fine-tuned from the $\pi_{0.5}$ [134] base model. Each fine-tuning run takes approximately 24 hours on 8xH100s. All of our real-world inference is done on a single NVIDIA RTX 4090 GPU in a workstation in the same building as the robots.

Appendix D

Appendices for Chapter 5

Algorithm 4 Python code implementing the loss and sampling functions for training-time action conditioning. Differences from standard flow matching code are highlighted in red.

```
import jax
import jax.numpy as jnp

def compute_loss(rng, model, observation, action_chunk, max_delay):
  b, ah, ad = action_chunk.shape # (batch_size, action_horizon, action_dim)
  noise_rng, time_rng, delay_rng = jax.random.split(rng)
  time = jax.random.uniform(time_rng, (b,))
  noise = jax.random.normal(noise_rng, (b, ah, ad))
  # sample delays from some distribution of choice:
  # here, we use Unif[0, max_delay), as in our real-world experiments
  delay = jax.random.randint(delay_rng, (b,), 0, max_delay)

  # set time to 1.0 for the action prefix
  # time becomes shape (batch_size, action_horizon)
  prefix_mask = jnp.arange(ah)[None, :]  < delay[:, None]
  time = jnp.where(prefix_mask, 1.0, time[:, None])

  # compute the noisy action postfix and run the model
  x_t = time[:, :, None] * action_chunk + (1 - time[:, :, None]) * noise
  pred_v_t = model(observation, x_t, time)
  loss = (pred_v_t - (action_chunk - noise))**2

  # compute the loss on the postfix only
  postfix_mask = jnp.logical_not(prefix_mask)[:, :, None]
  loss = jnp.sum(loss * postfix_mask) / (jnp.sum(postfix_mask) + 1e-8)
  return loss

def sample_actions(rng, model, observation, action_prefix, delay, num_steps):
  # assume action_prefix is padded to (batch_size, action_horizon, action_dim),
  # but only the first delay actions are valid
  b, ah, ad = action_prefix.shape
  x_t = jax.random.normal(rng, (b, ah, ad))
  time = 0.0
  dt = 1 / num_steps
  prefix_mask = jnp.arange(ah)[None, :]  < delay

  for _ in range(num_steps):
    x_t = jnp.where(prefix_mask[:, :, None], action_prefix, x_t)
    time_masked = jnp.where(prefix_mask, 1.0, time)
    v_t = model(observation, x_t, time_masked)
    x_t = x_t + dt * v_t
    time = time + dt

  return x_t
```

Appendix E

Appendices for Chapter 6

E.1 Implementation Details

We provide implementation details for SuSIE and the baselines. Table E.1 documents goal sampling hyperparameters.

SuSIE implementation details

Image-editing diffusion model

We finetune InstructPix2Pix [47] using similar hyperparameters to the initial InstructPix2Pix training. We use the AdamW optimizer [202] with a learning rate of 1e-4, a linear warmup of 800 steps, and weight decay of 0.01. We track an exponential moving average (EMA) of

	CALVIN	Smthn-Smthn	Bridge
k_{min} (Eq. 6.1)	20	11	11
k_{max} (Eq. 6.1)	22	14	14
k_δ (Eq. 6.2)	2	N/A	6

	CALVIN	Real-world
k_{test} (Alg. 2)	20	20

Table E.1: **(Goal sampling hyperparameters)** The first two hyperparameters, k_{min} and k_{max}, are used for training the image-editing model. k_δ is used for training the low-level policy. All 3 of these correspond to a particular dataset. k_{test} is a test-time hyperparameter that corresponds to an evaluation setup.

the model parameters with a decay rate of 0.999 and use the EMA parameters at test time. We train for 40k steps with a batch size of 1024 on a single v4-64 TPU pod, which takes 17 hours.

For simulation, we train the model exclusively on the CALVIN dataset. For real-world experiments, we cotrain with a sampling mixture of 60% Something-Something and 40% BridgeData V2.

At test time, we use an image guidance weight of 2.5 and a text guidance weight of 7.5. We use the DDIM sampler [299] with 50 sampling steps.

Goal-reaching policy

We use a diffusion model for our goal-reaching policy since recent work has shown that diffusion-based policies can better capture multi-modality in robot data [62, 114], leading to improved performance across a variety of tasks. In our implementation (which follows Walke et al. [329]), the observation and goal image are stacked channel-wise before being passed into a ResNet-50 image encoder. This image encoding is used to condition a diffusion process that models the action distribution. The diffusion process uses an MLP with 3 256-unit layers and residual connections. Following Chi et al. [62], rather than predicting a single action, we predict a sequence of 4 actions to encourage temporal consistency. We use the Adam optimizer [162] with a learning rate of 3e-4 and a linear warmup of 2000 steps. We train with a batch size of 256 for 445k steps on a single v4-8 TPU VM, which takes 15 hours. We augment the observation and goal with random crops, random resizing, and color jitter. For goal sampling, we use $k_\delta = 6$, meaning we sample goals uniformly from the range $[0, 20)$, since the corresponding $k_{\max}$ used with BridgeData V2 is 14 (see Table E.1).

At test time, we have several options for how to predict and execute action sequences. Chi et al. [62] use receding horizon control, sampling k-length action sequences and only executing some of the actions before sampling a new sequence. This strategy can make the policy more reactive. However, we found that the robot behavior was quite jerky as the policy switched between different modes in the action distribution with each sample. Instead, we use a temporal ensembling strategy similar to Zhao et al. [375]. We predict a new 4-action sequence at each timestep and execute the dimension-wise mean of the last 4 predictions for that timestep.

Baseline implementation details

MOO

MOO [306] utilizes a mask to represent the target objects and incorporates it as an additional channel in the observation. Specifically, they train a language-conditioned policy that takes a

4-channel image and a language command as inputs. To acquire the mask for target objects, the OWL-ViT [229] object detector is employed. This detector is an open-vocabulary object detection model, pretrained on Internet-scale datasets, and it is used to extract the bounding boxes of the objects of interest from the image. For tasks like "move X to Y," MOO extracts the bounding box for X, representing the object of interest, and Y, indicating the target place. A mask is then created where the pixel at the center of the predicted bounding box is assigned a value of 1.0 for X and 0.5 for Y.

Extracting object entities from BridgeData V2 language annotations. In order to obtain the mask, it is necessary to extract the entities corresponding to the object of interest, denoted as X, and the target place, Y, from the language command. In the MOO paper, the authors assume that the language in their dataset is structured in a way that facilitates easy separation of X and Y. Specifically, they employ a dataset that exclusively consists of language annotations such as "pick X," "move X near Y," "knock X over," "place X upright," and "place X into Y."

Given that the language annotations in BridgeData V2 are diverse and unstructured, it is challenging to naively extract X and Y. We utilized OpenAI's `gpt-3.5-turbo-instruct` model [39] to extract the object of interest and the target place (if any) from the language annotation and input them into OWL-ViT to create masks. It is worth noting that due to the unstructured nature of BridgeData V2, the extracted bounding boxes in training data are sometimes incorrect as shown in Figure E.1. We then trained a mask-conditioned LCBC policy using the same architecture as described in Appendix E.1. Following MOO, we removed X and Y from the prompt and replaced the entity X with "object of interest" and the entity Y with "target place". For example, given the language prompt "put the eggplant in the pot", we use a modified prompt "put object of interest in target place" as the input to the policy during both train and test time.

Test time. During test time, we manually give ground-truth masks to the policy. To enable this, we build a simple interface on the robot machine, allowing the evaluator to create the masks by clicking on the initial camera image at the beginning of each trial.

UniPi

UniPi [82] trains a video diffusion model $p_\theta(\tau \mid \mathbf{s}_0, l)$ to generate a sequence of frames given a language command and an initial frame. The original paper employs the model architecture from Imagen Video [124, 125]. To achieve higher resolution and longer videos for their real-world results, the authors leverage a 1.7B 3D UNet and four pretrained super-resolution models from Imagen Video, with 1.7B, 1.7B, 1.4B, and 1.2B parameters, respectively. Since the original models and code are not publicly available, we tried to replicate their approach in two different ways.

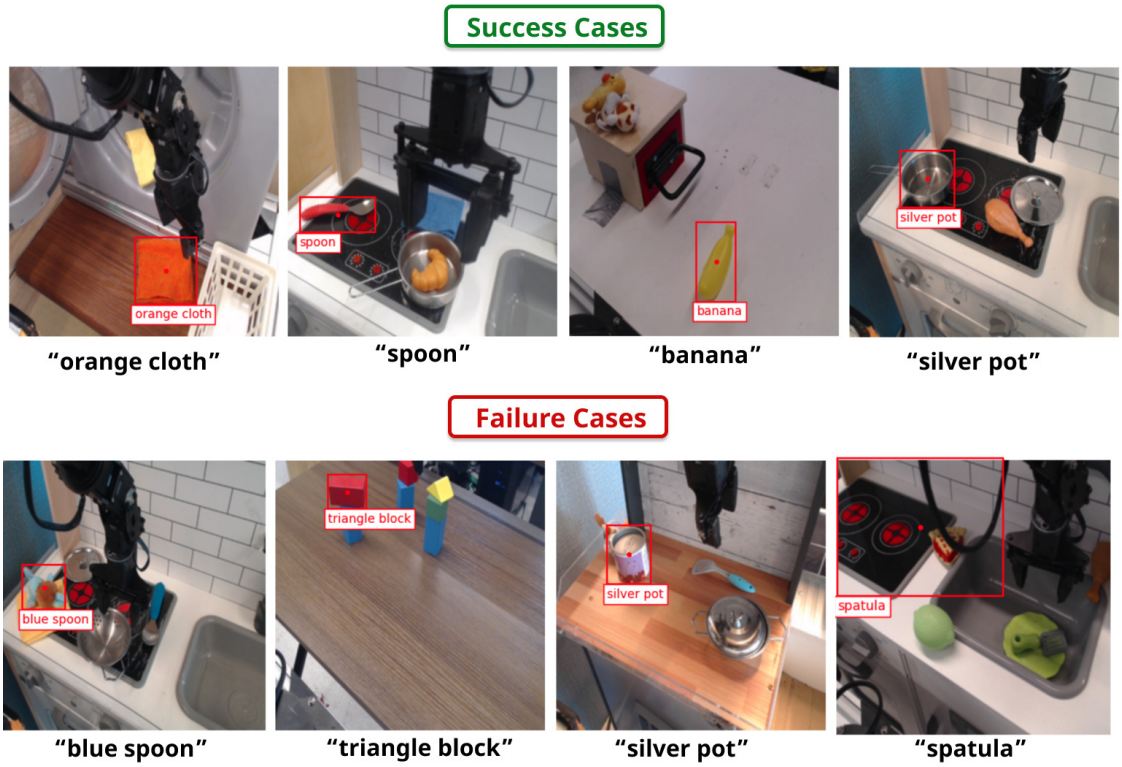


Figure E.1: **(Examples of extracted bounding boxes for MOO)**. Despite our best attempts, the bounding boxes extracted at train time are noisy due to the unstructured nature of BridgeData V2. This might have led to MOO's poor performance in our evaluations.

UniPi (HiP, Ajay et al. [6]) For the second approach, we followed the UniPi replication in Ajay et al. [6]. We trained a latent video diffusion model from PVDM [360], building upon the codebase `https://github.com/sihyun-yu/PVDM` where we added first-frame conditioning. We first trained the video autoencoder to project a video of size $16 \times 128 \times 128$ into a latent space, followed by training a PVDM-L model that uses a 2D UNet architecture. We used a Flan-T5-Base [64] encoder to obtain the language embeddings.

UniPi (Ours). We implemented a 3D UNet video diffusion model, following Ho et al. [125, 124] and adding UniPi's first-frame conditioning. Due to limited compute, we did not train spatial/temporal super-resolution models; instead, we trained a 3D UNet-based diffusion model to directly generate images with a resolution of 128×128. The model includes 4 residual blocks, with (input channels, output channels) as follows: $(64, 64)$, $(64, 128)$, $(128, 256)$, and $(256, 640)$. The model is trained to produce the trajectory with a fixed horizon of 10 frames $\tau_t = \{s_t, s_{t+1}, \ldots, s_{t+9}\}$ conditioned on the current frame s_t and language command. We used a frozen pretrained CLIP [264] encoder to obtain the language embeddings.

Data and training details. Following the UniPi baseline from HiP [6], we utilized Ego4D [103] to instill the model with Internet-scale knowledge. For UniPi (Ours), we first pretrained the video model on Ego4D for 270K steps, and finetuned it on the robotics dataset (CALVIN for simulation and BridgeData V2 for the real world) for an additional 200K steps. We used a batch size of 4 during training. For UniPi (HiP), we jointly trained a single model on Ego4D, BridgeData V2, and CALVIN at the same time. The autoencoder was trained for 85K steps, and the PVDM-L model was trained for 200K steps. We used a batch size of 8 during training.

Inverse model and test time control. To extract actions from generated videos, we trained a Gaussian inverse dynamics model $\pi_\phi(\mathbf{a} \mid \mathbf{s}_t, \mathbf{s}_{t+1})$ to predict the action from two adjacent frames. The model consists of a ResNet-50 image encoder and 3 256-unit MLP layers. During test time, given the current observation s_t and the language command l, we synthesize H image frames from the video model and apply the inverse dynamics model to obtain the corresponding $H-1$ actions. The predicted actions are executed, and we generate a new video from s_{t+H-1} and repeat the process.

Generated videos. While the quality of the video model trained on the simulation dataset is good enough for solving the tasks on the CALVIN benchmark as shown in Table 6.1, we found that it is nontrivial to obtain high-quality generations for the real-world scenes. We show examples of generations in Appendix E.3.

Language-conditioned behavior cloning (LCBC)

For the LCBC baseline, we use the same architecture and hyperparameters as the low-level policy in SuSIE. We encode the language instruction using MUSE [353] and feed it into the ResNet-50 image encoder using FiLM conditioning [251]. We found that this policy outperformed the LCBC policy used in Walke et al. [329] and Myers et al. [231] (which was not a diffusion policy), as well as the RT-1 architecture [42] trained on BridgeData V2 (also appearing in Walke et al. [329]).

E.2 Experimental Setup

CALVIN. We follow the evaluation setup from Mees et al. [226]. During each trial, the policy gets 360 timesteps total to complete a chain of 5 instructions, only moving on to the next instruction once the previous one is completed. Subsequent instructions in each language chain are chosen by the simulator based on environment affordances after completing the previous language instruction. Results in Table 6.1 are averaged over 100 trajectories, enabling direct comparison with results from Ge et al. [99], which are also averaged over 100 trajectories. Results for UniPi (HiP) and UniPi (Ours) were averaged over 25 trajectories,

due to the computational cost associated with querying the video diffusion models, and the relatively short video generation horizon (10 timesteps) requiring frequent regeneration.

To obtain comparisons with an oracle GCBC policy in Table 6.3, we sampled ground truth goal image states from the simulator to feed to the oracle policy by manually resetting the environment internal state to the state corresponding to the goal being achieved. This manual resetting process was only amenable for certain tasks in CALVIN, namely the ones in Table 6.3. Results in the table correspond to a total of 200 evaluation trajectories.

Real world. Figure 6.2 provides an exhaustive list of all 9 tasks we evaluate in the real world. We conduct 10 trials per task and report the average success rates. We vary the positions and orientations of various objects between trials, but use the same set of positions and orientations for evaluating all methods. In Scene A, we change the positions and orientations of the plate, eggplant, and carrot every trial; the pot always remains in the drying rack. For "put carrot on plate", we start the carrot in the pot for 4 out of 10 trials, and in the sink otherwise. The eggplant always starts in the sink. In Scene B, we change the positions of the bell pepper, orange pot, and ceramic bowl every trial, performing 5 trials where the target container is closer to the target object and 5 trials where it is further away. In Scene C, for "put X in the wooden bowl," we change the positions of the 3 objects every trial while the wooden bowl remains on the left; we also randomize the rotations of the objects up to 45 degrees from vertical. For "move the wooden bowl to the top of the table," we change the horizontal position of the bowl every trial, but it always starts at the bottom of the table.

E.3 Qualtitative Examples of Video Generation

Figure E.2 shows the video frames generated by the UniPi (HiP) model. The model fails to generate the robot's gripper accurately and consistently produces blurry results. For this reason, we did not evaluate this model on the real robot. Instead, we evaluated UniPi (Ours) model for the real-world tasks. Figure E.3 shows examples of success and failure cases of the test trials. While the robot succeeded in a few trials, we observed that in most cases the generated videos suffered from hallucination and physical inconsistency, which led to UniPi's poor performance.



Figure E.2: **(UniPi (HiP) example generations)**. Observe that the model is not able to generate a clear robot gripper, and produces videos that are blurry and visually poor.

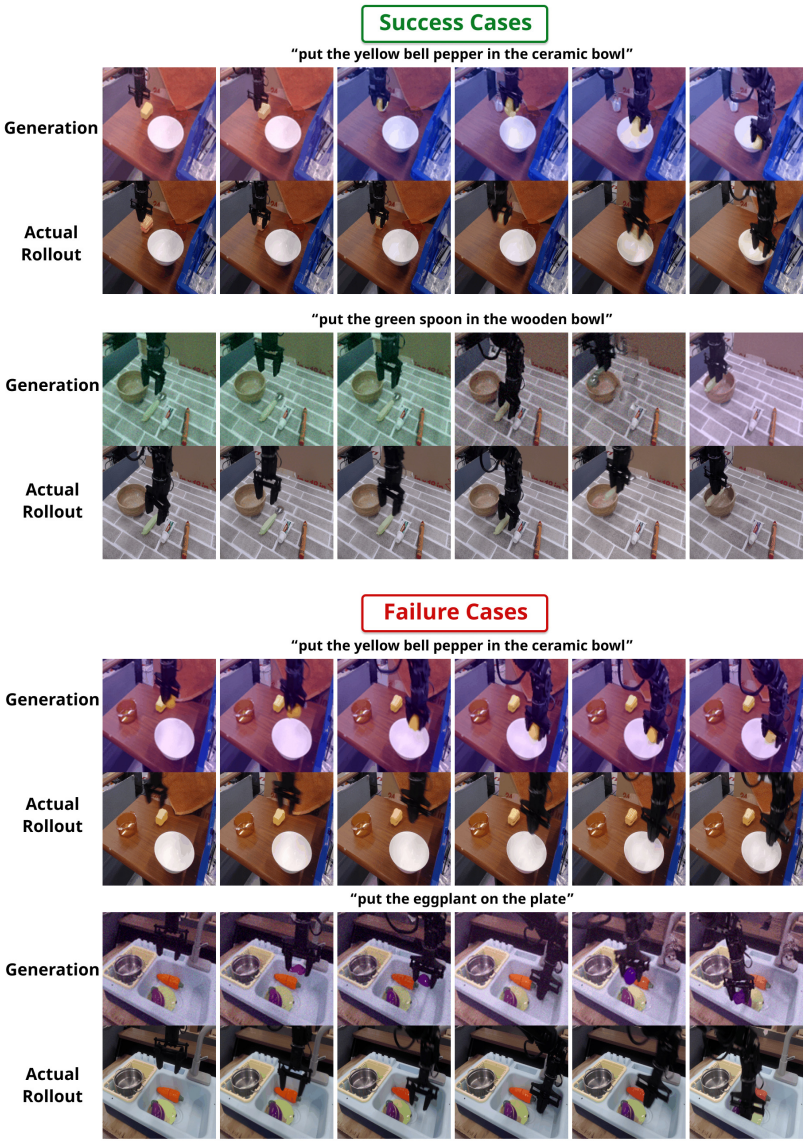


Figure E.3: **(UniPi (Ours) example generations and the resulting robot behaviors)**. While the robot succeeded in a few trials, we observed that in most cases the generated videos suffered from hallucination and physical inconsistency, which confused the low-level policy.

E.4 Qualitative Examples of SuSIE on CALVIN

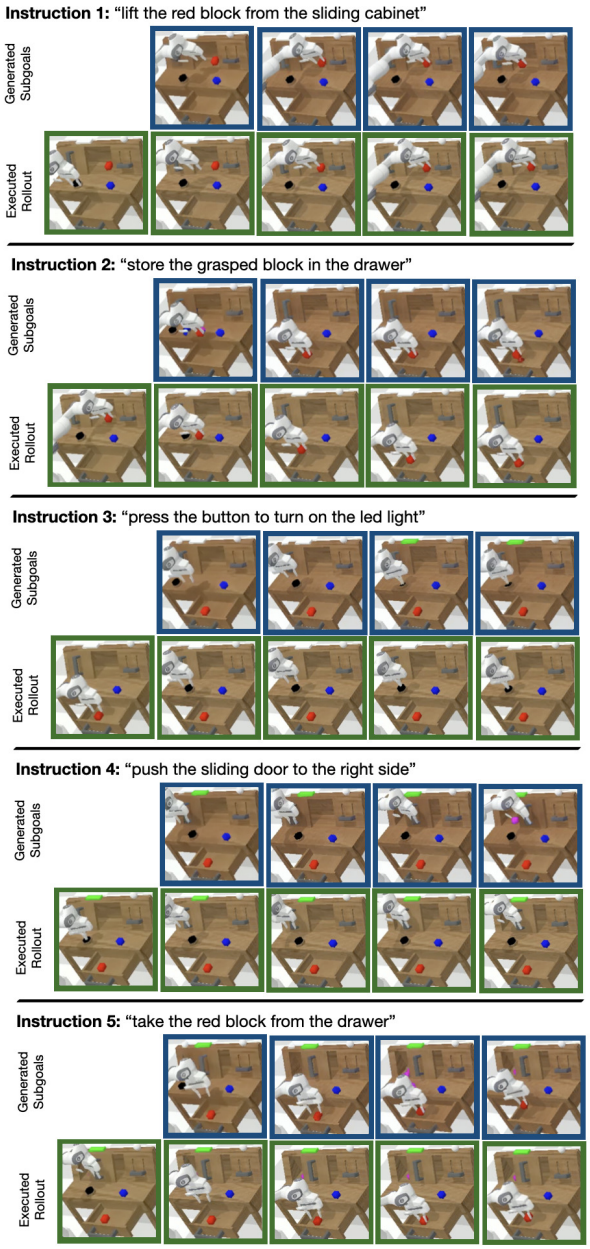


Figure E.4: **(Example rollouts in CALVIN.)** Visualization of SuSIE-generated subgoals and trajectory rollouts on a successful 5-instruction language chain in CALVIN.

Appendix F

Appendices for Chapter 7

F.1 Contributions

Data collection and operations. Ashwin Balakrishna, George Bokinsky, Thomas Charbonnier, Grace Connors, Michael Equi, Chelsea Finn, Lachlan Groom, Hunter Hancock, Karol Hausman, Connor Jacobsen, Rowan Jen, Marinda Lamb, Vishnu Mano, Nandan Marwaha, Aikys Mongush, Tyler Patterson, Charvi Sharma, Lucy Xiaoyang Shi, Laura Smith, Will Stoeckle, Anna Walling, Jason Wang, Samuel Whitmore, Blake Williams.

Annotation and supplemental data.. Ashwin Balakrishna, Karan Dhabalia, Danny Driess, Chelsea Finn, Haroun Habeeb, Rowan Jen, Chandra Kuchi, Karl Pertsch, Lucy Xiaoyang Shi, Will Stoeckle, Quan Vuong.

Policy training and research.. Bo Ai, Ashwin Balakrishna, Kevin Black, Danny Driess, Michael Equi, Yunhao Fang, Chelsea Finn, Catherine Glossop, Haroun Habeeb, Karol Hausman, Gashon Hussein, Victor Hwang, Brian Ichter, Liyiming Ke, Sergey Levine, Xinyu Li, Yao Lu, Suraj Nair, Karl Pertsch, Allen Z. Ren, Baifeng Shi, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, Jiaming Tang, Marcel Torne, Kyle Vedder, Quan Vuong, XuDong Wang, Charles Xu, Lili Yu, Wuming Zhang, Zhuoyang Zhang.

Policy infrastructure.. Kevin Black, Karan Dhabalia, Danny Driess, Mairbek Khadikov, Chandra Kuchi, Adrian Li-Bell, Vladislav Lialin, Wallace Lim, Yao Lu, Allen Z. Ren, Lucy Xiaoyang Shi, Kyle Stachowicz, Jiaming Tang, Quan Vuong, Haohuan Wang, Ury Zhilinsky.

Robot hardware.. Ali Amin, Raichelle Aniceto, Greg Balke, Vedant Choudhary, Foster Collins, Grace Connors, Maitrayee Dhaka, Adnan Esmail, Thomas Godden, Ivan Goryachev, Tim Jones, Gregg Kammerer, Ben Katz, Devin LeBlanc, Brendon LeCount, Zhonglin Liang, Enyu Luo, Liam Murphy, Gavin Schelske, Shalom Tekeste, Chris Whalen, Sukwon Yoo.

Robot infrastructure. Greg Balke, Kevin Black, Shihao Cao, Ken Conley, James Darpinian, Jared DiCarlo, Hunter Hancock, Karol Hausman, Szymon Jakubczak, Jimmy Tanner.

Writing and illustration.. Bo Ai, Ashwin Balakrishna, Kevin Black, Chelsea Finn, Sergey Levine, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Kyle Stachowicz.

F.2 Attention pattern

We describe the attention pattern used in training $\pi_{0.7}$ and the lightweight world model (for subgoal image generation), as well as running inference with them, in Fig. F.1.

F.3 Training of the world model

Our world model is initialized from BAGEL [72], and largely uses the same training recipe. We use a subset of our robot data and egocentric human video data with high-quality segmented language labels, as we found that label quality (especially temporal segmentation quality) has a large impact on subgoal quality. We additionally mix in several open-source image editing datasets and open-source video datasets to better preserve the semantic knowledge of the model. Each training example consists of a subtask instruction, $\hat{\ell}$, 3 camera inputs, $\mathbf{o}_t$, and 3 target images, $\mathbf{o}_{t_{\text{end}}}$, where t_{end} is the last timestep of the segment spanning t. Following BAGEL, the camera inputs are processed using both a ViT (for semantic understanding) and VAE (for fine-grained image details). The ViT tokens are further processed by a 7B LLM backbone, and the VAE tokens are processed by a 7B generation backbone. ViT inputs are resized to a resolution of 448x336, and VAE inputs (including target images) are resized to a resolution of 512x384. The discrepancy is due to the differing patch size of the ViT and VAE (14 and 16, respectively). At test time, we set $\Delta = 4$ seconds as the time interval for regenerating subgoals to match SuSIE [35].

F.4 Inference speed and optimization

The $\pi_{0.7}$ model and the high-level policy, which are both based on Gemma3 4B, use a single NVIDIA H100 GPU for inference. Various optimizations implemented after RTC [29] have brought the inference time of the minimal variant of $\pi_{0.7}$ down to 38ms with 3 camera inputs, 5 denoising steps, and training-time RTC [34] (which, unlike test-time RTC, incurs no additional inference-time overhead). Enabling the `MEM` vision encoder [322] and adding subgoal images to the context both incur additional overhead, bringing the inference time to 127ms in the worst case.

Generating subgoal images at a reasonable latency is challenging, due to the computational expense of iterative denoising with a 14B model and total sequence length of nearly 10,000

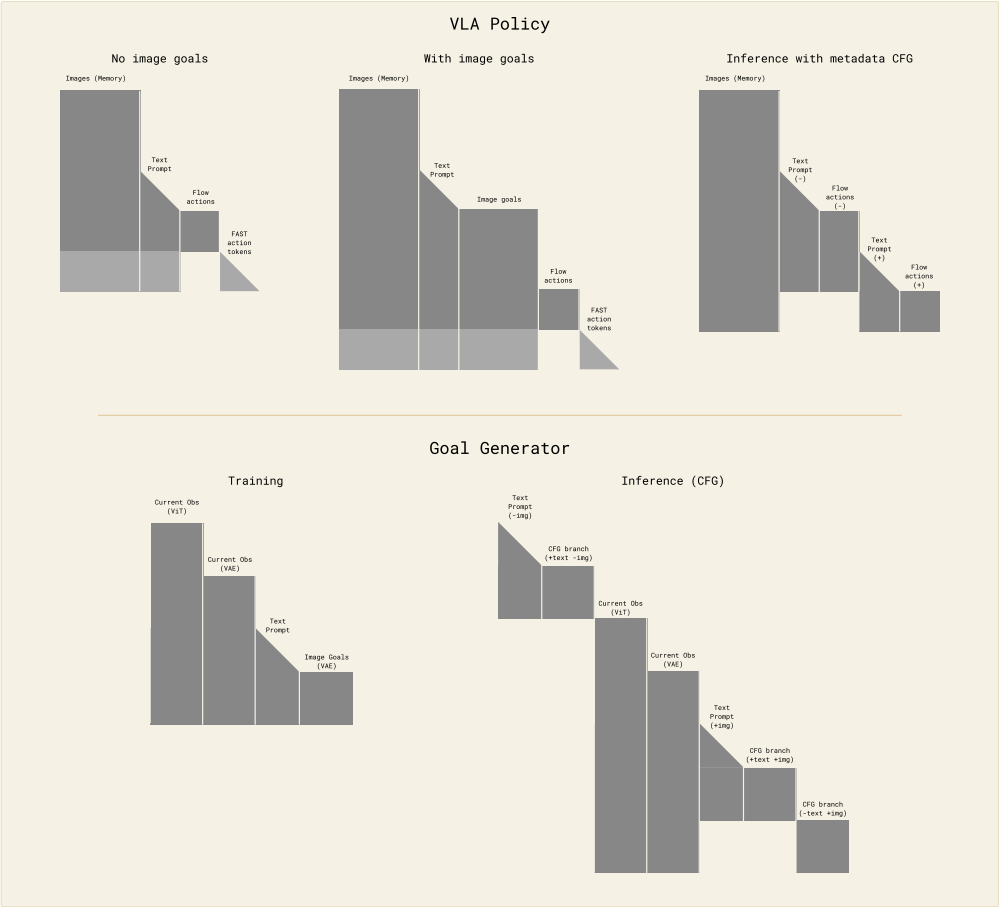


Figure F.1: The $\pi_{0.7}$ model and its world model (for generating subgoal images) use several different nontrivial attention patterns during training and inference. From top left: in absence of image goals we use the same attention patterns as in $\pi_{0.5}$, with global bidirectional attention between embeddings for all memory-aware image views. Note that the FAST tokens (only available at training time) and the flow actions do not attend to each other. When image goals are present, we include them as an additional block-causal bidirectional block, after the text prompt. When we do classifier-free guidance (at inference time), we pack both the positive and negative example into the same sequence for efficient inference by constructing an "attention tree" with two branches (positive and negative), which do not attend to one another. Following BAGEL, at training time the world model receives three copies of the image, each of which is block-bidirectional within the multiview group: current observation, encoded with ViT; current observation encoded with VAE; noisy image goal encoded with VAE. The world model uses a similar inference-time CFG trick, albeit with a more complex mask because it has three CFG groups rather than two.

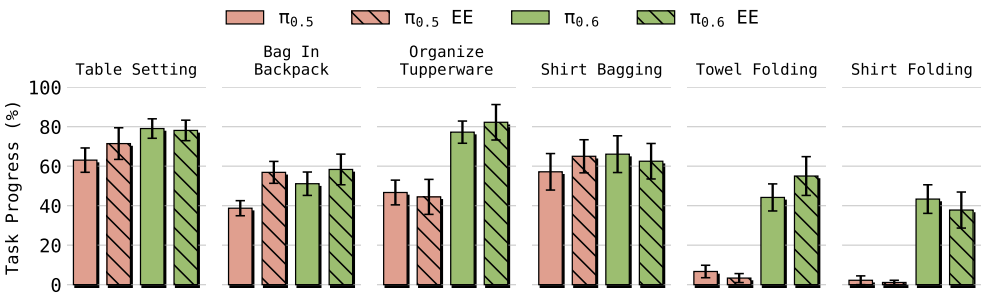


Figure F.2: **Joint vs. end-effector control for prior models on cross-embodiment tasks.** We compare joint-space and end-effector (EE) control for baseline policies across a range of tasks, observing no substantial difference in performance between the two control modes.

tokens. In addition to the aforementioned optimizations, we also use 4-way tensor parallelism on 4xH100 GPUs, quantize all large matrix multiplications to 8-bit precision, and use a modified version of SageAttention [369] for the backbone attention operations. This allows us to generate subgoal images with 25 denoising steps (each including both text and image CFG) in 1.25 seconds. At inference time, we execute using a naive asynchronous strategy, meaning that $\pi_{0.7}$ continues executing while the world model generates the next subgoal.

F.5 Comparison of action spaces in cross-embodiment transfer

Figure F.2 compares joint-space and end-effector (EE) control on cross-embodiment tasks. Across tasks, EE control does not show clear advantage. Therefore, we focus on joint-space control in the main cross-embodiment experiments (Sec. 7.9) for clarity.

F.6 Human subject study for cross-embodiment shirt folding

The human subject study serves two purposes. First, it provides the strongest possible baseline for evaluating $\pi_{0.7}$ by measuring how well expert operators can teleoperate the UR5e bimanual robot to perform the same task. Second, the result potentially motivates for cross-embodiment transfer: UR5e, an industrial manipulator with high joint inertia, is difficult to teleoperate precisely for dexterous tasks like shirt folding, making demonstration collection on this platform difficult. A model that transfers to new embodiment opens up opportunities to collect autonomous data without human teleoperation.

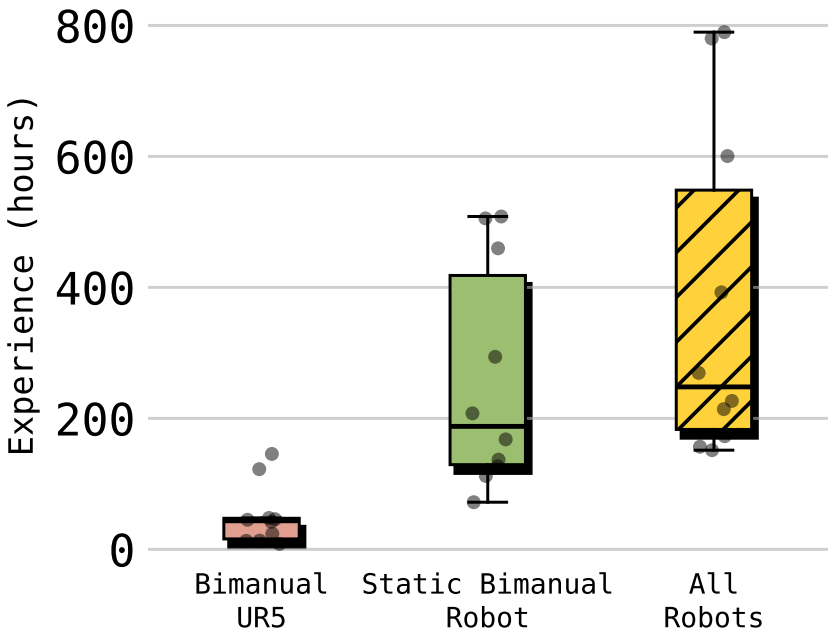


Figure F.3: **Operator experience in the human subject study.** Box plots show teleoperation experience (in hours) of the ten recruited operators across three categories: UR5e (target robot), the static bimanual robot (source robot), and all robots combined. The selected operators rank within the top 2% of our operator fleet in terms of teleoperation experience.

Participant selection. We recruited ten operators with the top 2 percentile of experience from the entire group of operators. They all have extensive prior experience teleoperating the source static bimanual robot, with an average experience of ∼375 hours across all robot platforms (Fig. F.3). Crucially, none had prior experience performing the shirt folding task on the UR5e, mirroring the "zero-shot" setting of our learned policy.

Protocol. Each operator performed three trials, yielding 30 total trials. To match the zero-shot transfer setting of $\pi_{0.7}$, operators received no practice or warm-up period before their first attempt. The initial shirt configuration (flattened on the table), the time limit, and the evaluation criteria were all identical to those used in the $\pi_{0.7}$ policy evaluation. Operators were instructed to maximize task success to align with the evaluation metric. We report task progress and success rate using the same metrics as in our robot experiments, ensuring a direct and fair comparison.

Results. Fig. F.4 compares $\pi_{0.7}$ (GC) with human teleoperators under the same zero-shot setting. Human operators achieve an average task progress of 90.9% and a success rate of

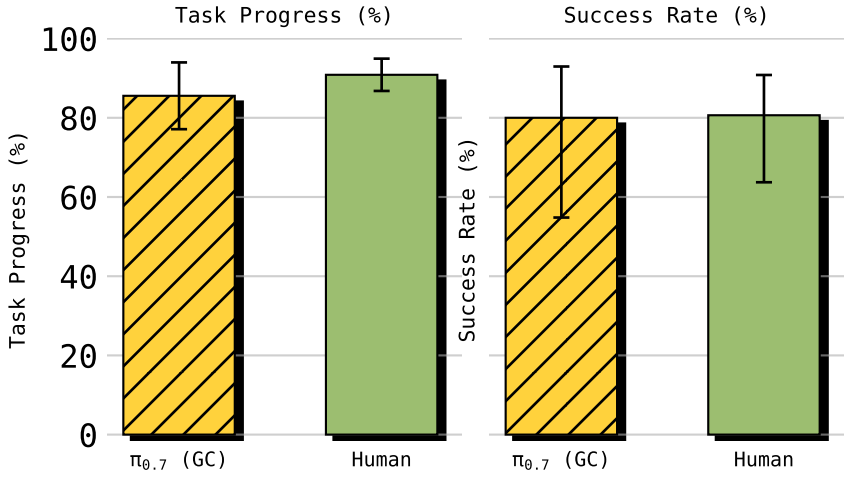


Figure F.4: **Comparison of $\pi_{0.7}$ (GC) and human.** We find that $\pi_{0.7}$ (GC) achieves competitive performance compared to the human operators, in the shirt folding task with the UR5e bimanual platform.

80.6%. $\pi_{0.7}$ achieves 85.6% task progress and an 80% success rate, demonstrating performance comparable to these expert operators despite not trained with any folding data from the UR5e platforms. These results provide strong evidence of zero-shot cross-embodiment transfer in $\pi_{0.7}$.

F.7 Detailed task descriptions and scoring rubric

Laundry (T-shirts and Shorts). Fold a single pair of shorts or a T-shirt taken from a laundry basket into a compact, neatly aligned fold and place/stack it appropriately. Scoring: success if successfully folding the item and properly stacking it on top of other laundry.

Laundry (Diverse — Hardest Item). Fold a single buttonup shirt taken from a laundry basket into a compact, neatly aligned fold and place/stack it appropriately. Scoring: success if successfully folding the item and properly stacking it on top of other laundry.

Make Espresso. Execute an espresso workflow to prepare and extract a doppio (double espresso) with correct sequencing and placements. Scoring: success if completing the required steps in order: grounds dispensed and tamped, portafilter locked into the correct grouphead, extraction performed, cup placed on saucer and moved right.

Box Building. Assemble a flat box into a 3D box. Scoring: success if the box is folded correctly without major damage.

Make Peanut Butter sandwich. Make a peanut-butter sandwich, cut it diagonally all the way through, and present it on the plate; close the peanut butter jar; push the plate away. Maximum score: 9. Points awarded for: removing the jar lid; progressively spreading peanut butter coverage; placing the plain slice on top; cutting diagonally fully through; placing the knife back on the plate; pushing the plate away; and overall neatness.

Turn a T-shirt Inside Out. Retrieve an inside-out t-shirt, turn it fully right-side-out (torso + both sleeves), fold it into a compact form. Maximum score: 7. +1 retrieve shirt; +1 torso right-side-out; +1 left sleeve right-side-out; +1 right sleeve right-side-out; +1 compact fold; +1 placed on pile; +1 positioned in upper-right corner.

Drive Through Door. Open a self-closing closet door, drive the robot fully inside, and finish with the door closed and the robot inside. Maximum score: 3. +1 for opening the door; +1 for driving far enough inside for the door to close; +1 for smooth entry without bumping into the doorway.

Cut Zucchini. Slice a zucchini into thin slices using a knife attached to a lanyard while stabilizing the zucchini with the opposite gripper. Maximum score: 3. +1 for successfully picking up the knife correctly; +1 for fully cutting the zucchini into thin, even slices; +1 for safely returning the knife to the right side of the cutting board.

Peel Fruits and Vegetables. Peel a fruit/vegetable completely by holding the item against cutting board, and peeling it with the other gripper. Maximum score: 9. +1 for picking up the peeler; +1 for peeling up to 25%; +1 for peeling 25–50%; +1 for peeling 50–75%; +1 for peeling >75%; +1 for putting the fruit/veg into the bowl; +1 for scraping 25–50% of food scraps into the trash can; +1 for scraping 50–75% of food scraps into the trash can; +1 for scraping >75% of food scraps into the trash can.

Take Out Trash. Remove an old trash bag, place it away from the bin, line the trash can with a new bag, and return the bin to its home location (and close the cabinet / replace lid as appropriate). Maximum score: 12. Taking out trash can (3 points): +2 if the robot correctly accesses the trash can (opens the under-sink cabinet door); +1 if the robot relocates the trash can from under the sink to an accessible position on the kitchen floor. Remove from bin (3 points): +1 if the robot pulls the trash bag away from the corners of the bin; +1 if the robot gathers and securely lifts the trash bag out of the bin; +1 if the robot removes the trash bag from the bin and places it on the floor near the trash can. Replace bag (3 points): +1 if the robot picks up the replacement trash bag; +1 if the robot opens the replacement bag; +1 if the robot places the replacement trash bag fully inside the trash can and stretches the edges securely around the rim. Return trash can back (3 points): +2 if the robot lifts and returns the trash can to the under-sink cabinet; +1 if the robot closes the cabinet door fully at the end of the episode.

Swap 3 Mugs. Sequentially place three mugs onto a coffee machine drip tray (one at a time), ensuring each mug occupies the tray once, with mugs returned to the table between placements as required. Maximum score: 4. +1 for removing mug 1 from the coffee maker and placing it on the table; +1 for placing mug 2 from the table onto the coffee maker; +1 for removing that same mug 2 from the coffee maker and placing it on the table; +1 for placing the remaining mug 3 from the table onto the coffee maker.

Find Object. Retrieve an object hidden in a drawer and restore a clean final scene (object on table, drawers closed). Maximum score: 4. +1 for fully opening the target drawer with the item on the first try; +1 for picking up the sponge stick; +1 for placing the sponge stick on the table (anywhere); +1 for closing the opened target drawer.

Scoop Beans. Scoop exactly two scoops of coffee beans into a grinder using a measuring cup, then close the grinder lid. Maximum score: 5. +1 for opening the coffee grinder; +1 for grasping the scoop; +1 for successfully collecting and pouring the first full scoop of beans into the coffee grinder; +1 for successfully collecting and pouring the second full scoop of beans into the coffee grinder; +1 for closing the coffee grinder lid.

Window Cleaning. Spray the phone booth window with windex, rip a paper towel, wipe the glass fully dry, and discard the towel. Maximum score: 5. +1 for spraying the booth; +1 for getting the paper towel for cleaning; +1 for cleaning overall the whole door and leaving it dry; +1 for throwing the paper towel in the trash can; +1 for not leaving any drops left on the door.

Reverse Bussing. Sort 12 objects with reversed mapping: place trash into the bussing bin and dishes/utensils into the trash can. Maximum score: 12. +1 for successfully putting a plate, cup, bowl, or utensil in the trash (7 total); +1 for successfully putting a plastic bottle, foil, plastic lid, take out container, or chip bag in the bussing bin (5 total).

Reverse Fridge to Microwave. Move a plate of (real) frozen microwavable food from the microwave into the refrigerator (reverse sequence version), and complete the episode with the plate stored in the fridge. Maximum score: 6. +1 for microwave door opening; +1 for microwave door closing; +1 for plate removed from microwave; +1 for plate placed in refrigerator; +1 for refrigerator door opening; +1 for refrigerator door closing.

Table Setting. Set a table by placing a placemat, cup, plates, napkin, and utensils from a bin into a reasonable table setting. Maximum score: 7. +1 per item successfully placed; critically incorrect placements can be scored as -1.

Bag in Backpack. Place a small pouch/bag into a backpack (no need to zip closed). Maximum score: 3. +1 pick up pouch; +1 grasp backpack; +1 place pouch inside backpack.

Organize Tupperware. Nest three tupperware containers (descending sizes) and stack their corresponding lids. Maximum score: 6. +1 for each container correctly nested; +1 for each lid correctly stacked/aligned.

Shirt Bagging. Place two shirts fully inside a brown grocery bag (per-scene), optionally using language commands with timing-sensitive instruction handoff. Maximum score: 4. +1 for correct pick for each shirt; +1 for each shirt fully placed inside the bag.

Shirt Folding. Fold a single t-shirt from a flattened start state into a folded end state. Maximum score: 6. +1 for completing the first fold (grasp cloth with both arms; folded edge aligned with intended fold within 5 inches); +1 for completing the second fold (same criterion); +1 for completing the final fold (completed with right arm; aligned within 5 inches); plus a fold quality score from 0–3 based on the final folded state chart. Full score of 6 is considered success.

Press French Press Plunger. Press the french press plunger fully down to the bottom. Maximum score: 1. +1 if the robot manages to press the plunger down to the bottom (through coffee).

Scoop Rice into Rice Cooker. Pick up a rice scooper from a rice container, scoop rice, and pour it into an open rice cooker. Maximum score: 3. +1 pick up scooper; +1 scoop rice; +1 pour into cooker.

Loading an Air Fryer. Open an air fryer, place a sweet potato into the basket, and close the air fryer. Maximum score: 4. +1 open air fryer; +1 pick up sweet potato; +1 place into air fryer; +1 close air fryer.

Unloading an Air Fryer. Pull out an air fryer basket and dump 8 fake fries onto a plate. Maximum score: 2. +1 if the robot successfully picks up all food items from the air fryer; +1 if the robot places all food items onto the plate.

Toast a Bagel. Place a sliced half-bagel into a toaster oven, initiate toasting by turning the knob, then retrieve and serve it on a plate taken from an overhead cabinet. Maximum score: 7 (open oven; insert bagel; close oven; turn knob; retrieve plate; retrieve toast; place on plate).