CARP: Visuomotor Policy Learning via Coarse-to-Fine Autoregressive Prediction

Supplementary Material

A. Limitations and Future Work

In this work, we propose CARP, a next-generation paradigm for robotic visuomotor policy learning, which effectively balances the long-standing trade-off between high performance and high inference efficiency seen in previous autoregressive modeling (AM) and diffusion modeling (DM) approaches. Despite these advancements, there remain several limitations and opportunities for improvement in near-future research.

First, the architectural design of CARP can be further optimized for simplicity. Currently, CARP employs a two-stage design, where the first stage utilizes separate *multi-scale* action VQVAE modules for each action dimension to address their orthogonality. A promising direction for future work could focus on developing a unified one-stage method that integrates *multi-scale* tokenization with the *coarse-to-fine* prediction process, resulting in a more efficient and streamlined framework without compromising performance.

Second, CARP's multimodal capacity has not yet been fully explored or leveraged. To address the inherent unimodality of conventional autoregressive policies trained with MSE loss, CARP employs a Cross-Entropy objective that preserves the potential for multi-modal predictions. Compared to the Diffusion Policy [7]'s ability to model multimodality via DDPM's integration over stochastic differential equations [10], CARP adopts a more direct yet effective alternative that achieves comparable multimodal expressiveness. Nevertheless, the role of multi-modality in visuomotor policy learning remains underexplored. Many current benchmark tasks either do not require diverse output distributions or tend to induce overfitting to a single prediction path. Future research should investigate the necessity of multi-modal reasoning in robotic decision-making and further harness CARP's capacity to model action diversity.

Third, CARP's adoption of the GPT-style paradigm opens up promising yet unexplored possibilities. Beyond the flexibility already demonstrated, the contextual understanding capabilities inherent in GPT-style architectures [21] suggest that CARP could be extended to support multi-modal inputs [2, 22] like tactile and auditory information and address robotic tasks requiring long-term dependency reasoning [1]. Moreover, its inherent capacity for in-context learning suggests strong potential for generalization under few-shot and zero-shot learning settings [5], making it a compelling foundation for more adaptive and versatile visuomotor policies.

Finally, but not exhaustively, the scaling potential of CARP presents a promising avenue for future exploration. The scaling laws established in existing GPT-style models [13] could be seamlessly applied to CARP, suggesting that increasing model capacity and leveraging larger pre-training datasets could lead to substantial performance gains. Furthermore, recent advances in Vision-Language-Action (VLA) [4, 14, 18] models present a promising opportunity to integrate CARP into such frameworks. Such integration could further demonstrate CARP's scalability and its potential for general-purpose embodied intelligence.

B. Coarse-to-Fine Inference

Unlike the training process, the inference process predicts token maps of the action sequence across different scales in an autoregressive *next-scale*, *coarse-to-fine* manner without teacher forcing, as illustrated in Fig. 1. Additionally, kv-caching is employed to eliminate redundant computations.

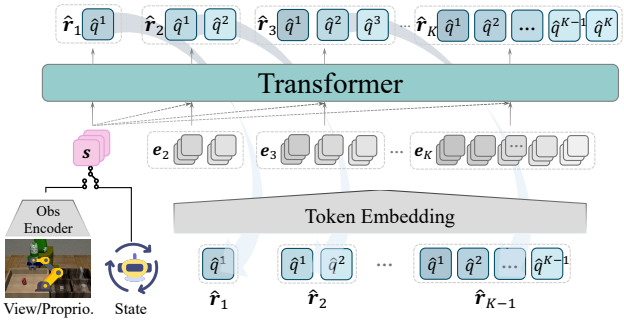

Figure 1. **Coarse-to-Fine Autoregressive Inference.** During inference, we leverage kv-caching to enable *coarse-to-fine* prediction without the need for causal masks. The full set of token maps, $\boldsymbol{r}_{1:K}$, is collectively decoded by the action *multi-scale* VQVAE into executable actions for the robotic arm.

C. Definition of Autoregressive Policy

Autoregressive policies naturally capitalize on the efficiency and flexibility of autoregressive models. Initial works from a reinforcement learning perspective applied models like Transformers to predict the next action using states or rewards as inputs [6, 11], as shown in Fig. 2 and formalized as

$$p\left(\boldsymbol{a}_t, \boldsymbol{a}_{t+1}, ..., \boldsymbol{a}_{t+H-1}\right) = \prod_{k=t}^{t+H-1} p\left(\boldsymbol{a}_k | \boldsymbol{a}_{k-H:k-1}, \boldsymbol{s}_{k-H:k}\right), \tag{1}$$

where $\boldsymbol{s}_{k-H:k-1}$ represents the states or observations corresponding to the previous actions $\boldsymbol{a}_{k-H:k-1}$, and $\boldsymbol{s}_k$ is the

current state or observation. Following this paradigm, several subsequent works [12, 23, 25] employ autoregressive models to predict one action at a time during inference.

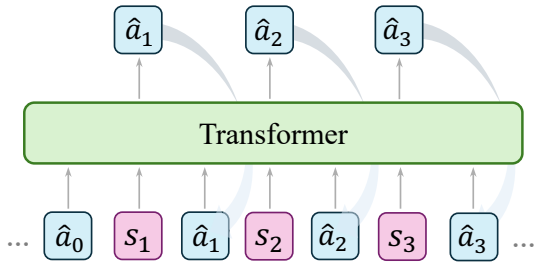

Figure 2. **Conventional Autoregressive Policy.** In reinforcement learning, conventional autoregressive policies generate action tokens sequentially, where each token is predicted based on the previously generated tokens. This differs from the action chunking prediction (see Sec. 2.2 of the main paper).

More recently, the concept of action chunking [15], derived from neuroscience, has demonstrated notable benefits for imitation learning [8, 17, 24, 28]. In action chunking, individual actions are grouped and executed as cohesive units, leading to improved efficiency in storage and execution, as depicted in Fig. 2a of the main paper. This paradigm extends the capabilities of GPT-style decoders by modifying them to generate chunks of actions in one forward pass, replacing the traditional single-step autoregressive operation with a multi-token, pseudo-autoregressive process. The action generation process for this paradigm is described as

$$p\left(\boldsymbol{a}_t, \boldsymbol{a}_{t+1}, \ldots, \boldsymbol{a}_{t+H-1}\right) = \prod_{k=t}^{t+H-1} p\left(\boldsymbol{a}_k | \boldsymbol{s}_O\right), \quad (2)$$

where O is the historical horizon. The model predicts the entire action sequence in one forward pass without strictly adhering to step-by-step autoregressive operations.

Given the significant performance improvements enabled by action chunking, we adopt this multi-token, one-forward-pass framework throughout the article and experiments when referring to Autoregressive Policy (AP).

D. Efficiency Concerns

Efficiency, as discussed throughout this paper, specifically refers to inference efficiency—the ability of CARP to generate actions significantly faster than diffusion-based policies during deployment. While efficiency can be examined from various perspectives, we focus on three key aspects relevant to CARP: inference efficiency, training efficiency, and data efficiency, each of which is analyzed in detail below.

Inference Efficiency. We analyze the inference efficiency of CARP's two-stage process. During inference, CARP first predicts action tokens in a *coarse-to-fine* manner, followed by a single forward pass to decode all token maps into executable actions. Since token maps are collected during the prediction phase, and decoding requires only a single forward computation, the majority of the computational cost is incurred during prediction, while action decoding remains relatively lightweight. This is empirically validated by the results in Tab. 1. Compared to DP, CARP achieves significantly faster inference by eliminating the iterative denoising steps required by diffusion-based policies, instead directly predicting actions as a low-dimensional generation problem.

Task	*CARP*		*DP*
	Predict	Decode	
Can	2.279 s	0.639 s	34.79 s
Square	2.597 s	0.679 s	35.62 s

Table 1. **Inference Efficiency Comparison.** We report the time consumption for the *coarse-to-fine* prediction phase and the subsequent action decoding phase over 400 timesteps of action generation. The results indicate that in CARP, the majority of inference time is allocated to the prediction step, whereas the decoding process is completed within a short duration.

Training Efficiency. We analyze training efficiency through convergence behavior and time consumption.

While training convergence depends on task complexity and hyperparameter configurations, both CARP and DP exhibit stable learning dynamics under our respective settings. To accommodate the architectural differences of CARP, we employ slightly different training configurations from those used for DP. As shown in Fig. 3, under our experimental settings, both DP and CARP achieve good convergence within the same number of training epochs. While convergence speed may differ due to structural and training differences, it does not inherently indicate superiority in model design. Instead, both CARP and DP demonstrate reliable training behavior under their respective settings.

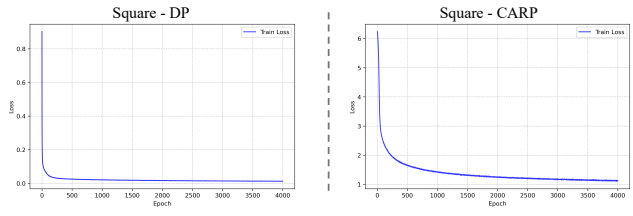

Figure 3. **Training Efficiency on Convergence Analysis.** With different training configurations, both DP and CARP converge effectively within 4000 epochs in the state-based Square task.

In terms of wall-clock training time, a comparison is provided in Fig. 4. The total training cost of CARP is comparable to that of DP when considering the policy learning stage (`CARP-TF`) alone. Although CARP introduces an additional tokenizer pretraining phase (`CARP-VQ`)—where separate

	GPU hours	GPU Memory	CPU Memory
DP-C	12.17 h	3.06 GB	18.13 GB
DP-T	10.83 h	1.56 GB	15.51 GB
CARP-VQ	4.67 h (28min×10)	0.69 GB	2.55 GB
CARP-TF	13.85 h	2.91 GB	19.86 GB

* A distinct VQ is trained for each dimension (10 total in our setting).
* Test on state-based Square with 200 trajectories using a V100 GPU.
* Similar results across simulation and real-world dataset.

Figure 4. **Training Efficiency on Time Consumption.** Comparison of training time on the Square task. For CARP, we separately report the tokenizer training time (with 10 VQ-VAEs, one per action dimension) and policy learning time.

VQ-VAEs are trained for each action dimension—the cost is amortized across tasks and environments. In particular, when trained on sufficiently diverse data (e.g., multi-task settings in both simulation and the real world), the tokenizers become reusable, substantially reducing the overall training burden in practical deployments.

Data Efficiency. We further assess the data efficiency of CARP by evaluating its performance under varying amounts of training data. Specifically, we investigate whether CARP can maintain strong performance when trained with limited trajectories, indicating robustness to data scarcity. As shown in Fig. 5, we compare CARP with baseline policies (following implementations from [7]) on the state-based Square task, using training datasets ranging from 200 to 30 trajectories. CARP consistently outperforms the baselines across all data regimes, demonstrating its superior data efficiency and reduced reliance on large-scale datasets.

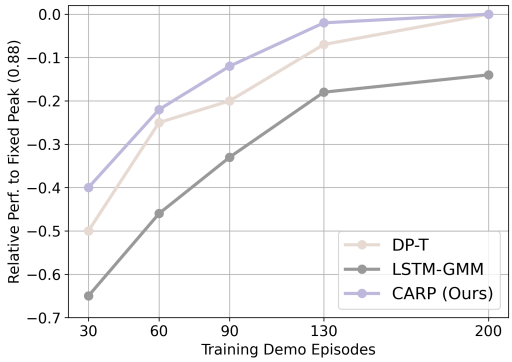

Figure 5. **Data Efficiency Analysis.** Performance comparison under varying training dataset sizes on the Square task. Each policy is trained following its official best-practice settings. CARP consistently outperforms the baselines at all data scales.

E. Comparative Analysis of CARP and AR.

CARP outperforms traditional autoregressive (AR) models in terms of success rate while maintaining high computational efficiency. CARP adopts a straightforward action tokenizer and leverages a GPT-style transformer for prediction, similar to standard AR policies. However, instead of conventional *next-token* prediction, CARP introduces a paradigm shift towards *next-scale* prediction. Despite these seemingly minor modifications, CARP achieves substantial performance gains. In this section, we analyze the key factors contributing to this improvement.

Temporal Locality. CARP encodes action sequences into a latent space in its first stage. Specifically, it employs 1D convolution to explicitly capture the local correlations within actions, facilitating a more effective learning of temporal dependencies—something that step-by-step action modeling struggles to achieve. As depicted in the magnified region of Fig. 7, encoding actions into a latent space enhances the smoothness of predictions while simultaneously denoising raw actions. This encoding process enables the model to capture similarities and overarching trends across contiguous actions, leveraging temporal locality to its advantage. While recent work on action chunking [28] has highlighted the significance of temporal locality, existing *next-token* prediction models still suffer from weakened action dependencies due to their traditional independent action output mechanisms.

Global Structure. CARP represents action sequences across multiple scales and predicts actions in a *coarse-to-fine* manner. The coarser scales compress the sequence using fewer tokens, promoting the learning of global action patterns. This hierarchical representation explicitly models the overall structure of action sequences—an aspect that traditional unidirectional *next-token* prediction struggles to capture. By progressively refining actions from high-level to low-level representations, CARP enhances sequence stability and mitigates the risk of producing erratic, inconsistent motions, leading to more precise execution.

Action Scalability. Similar to the approach used in Diffusion Policy, encoding action sequences improves the scalability of action generation. By encoding action sequences, CARP enables flexible adjustments to sequence length with minimal modifications to the model architecture, offering greater adaptability across different tasks and environments.

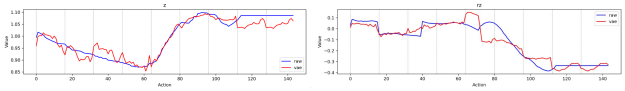

Figure 6. **Raw vs. Reconstructed Actions (Jointly Trained VQ-VAE).** Flattening and jointly encoding all action dimensions into a single VQ-VAE leads to poor reconstruction, as shown by the large discrepancy between raw (blue) and reconstructed (red) trajectories. This highlights the limitation of naive joint encoding.

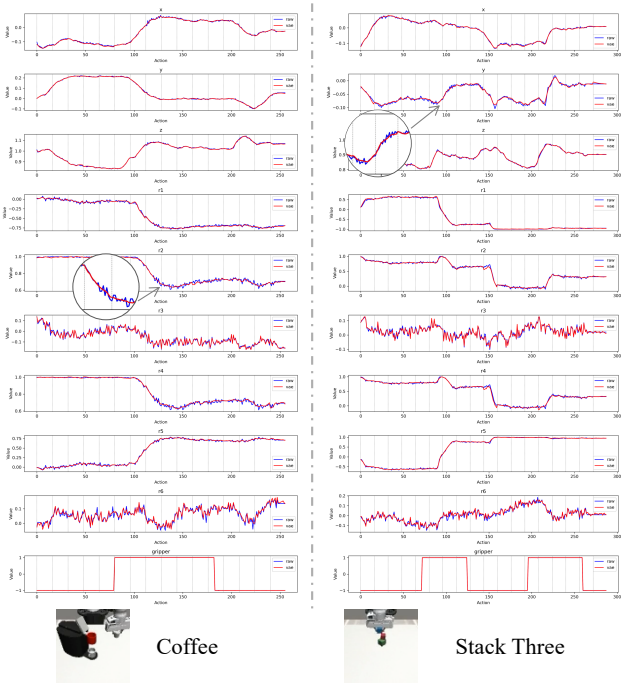

Figure 7. **Comparison of Raw and Reconstructed Actions.** Comparison across 10 action dimensions in the Coffee and Stack Three tasks. Reconstructed actions (red) closely align with raw signals (blue), preserving structural patterns while smoothing the sequences and filtering out noise (see magnified region), highlighting the effectiveness of our *multi-scale* tokenization.

Precision in Encoding. Adapting the *multi-scale* VQ-VAE to action space necessitates careful architectural design. Rather than flattening the action trajectory into an image-like structure and training a single joint VQ-VAE—which leads to unstable and less interpretable tokens (Fig. 6)—we instead encode each action dimension independently, using a dedicated VQ-VAE per dimension. As illustrated in Fig. 7, the *multi-scale* tokenization process ensures that generated action sequences closely match the raw inputs, exhibiting nearly identical trajectory lines while yielding smoother motions. This demonstrates that our action tokenization approach effectively preserves the fidelity of original action sequences. Moreover, the enhanced success rates observed in the experiments presented in the main paper further validate the accuracy and effectiveness of CARP's design.

F. Additional Baselines Comparison

In addition to the baseline policies discussed in the main paper, we further compare CARP with enhanced versions of each category: VQ-BET [17], an improved variant of BeT [24], and Consistency Policy [20], which reduces sampling steps to improve inference efficiency. Both are implemented using their official codebases with recommended settings. As shown in Tab. 2, CARP achieves consistently higher success rates across all tasks, while maintaining competitive inference time. These results highlight CARP as a promising design that combines strong task performance with high inference efficiency.

Policy	p1	p2	p3	p4	Inf.T(s)	Push-T	Inf.T(s)
ConsisP	0.99	0.96	0.95	0.93	2.31	0.80	2.93
VQ-BeT	0.96	0.92	0.87	0.71	1.48	0.72	1.70
CARP	**1.00**	**1.00**	**0.98**	**0.98**	2.01	**0.88**	2.66

Table 2. **State-Based Kitchen and Push-T Results.** We compare CARP with VQ-BET [17] and Consistency Policy [20] under identical settings. CARP consistently outperforms both baselines in success rate, while offering competitive inference times.

G. Failure Analysis

In this section, we analyze common failure cases observed during experiments with both the diffusion-based policies and our proposed CARP.

Accident Recovery. A notable failure mode is the inability to recover from disturbances, as illustrated in Fig. 8. When tools are accidentally knocked over due to suboptimal action trajectories, the model struggles to generate appropriate recovery behaviors. This limitation arises because the policy is trained purely by imitating expert demonstrations, which do not account for such out-of-distribution failure scenarios. Addressing this issue requires further incorporation mechanisms for failure detection and recovery.

Hesitant Movements. Another common failure case involves the generation of jerky or oscillatory movements when the robot encounters two similar situations with only slight visual differences across consecutive timesteps, as shown in the first row of Fig. 9. This issue arises because the policy conditions its predictions on only the previous one or two observations, potentially overlooking long-term historical context. When faced with multiple plausible action choices under limited observations, the policy may produce ambiguous actions, leading to hesitation. Consequently, these hesitant movements can prevent the robot from meeting the success criteria, as shown in the bottom-right of Fig. 9.

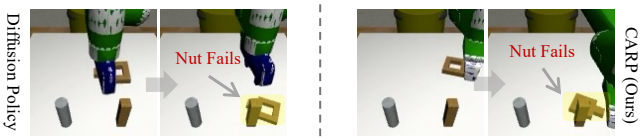

Figure 8. **Accident Recovery in the Square Task.** When tools are accidentally knocked over, the robot struggles to recover due to its reliance on imitation learning from expert demonstrations, which lack exposure to such out-of-distribution failure cases.

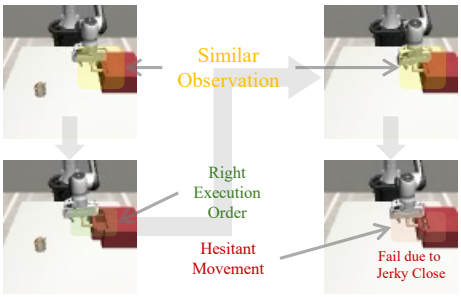

Figure 9. **Hesitant Movements in the Mug Task.** During task execution, the robot may encounter visually similar observations at different timesteps. Without leveraging long-term historical context, the policy may misinterpret these similarities and generate ambiguous actions, resulting in hesitant (jerky) movements. In this failure case, after successfully closing the drawer, the policy perceives the scene as similar to the initial step and erroneously attempts to reopen it. This cycle of opening and closing continues indefinitely, leading to task failure.

H. Experiment Implementation Details

Here, we provide implementation details for the main experiments presented in the paper.

Single-Task Simulation Experiment. For baseline models, we follow the same implementation and training configurations provided by Diffusion Policy. For all state-based experiments, including the Kitchen and Push-T tasks, we uniformly set the observation horizon $O = 2$ and the prediction horizon $H = 16$ across all models. For image-based experiments, we set $O = 1$ and $H = 16$ for better transferability to real-world scenarios. As per the benchmark, only the first 8 actions in the prediction horizon are executed, starting from the current step (see Suppl. I for further discussion). For CARP, we first train an action VQVAE model (see Sec. 3.1 of the main paper) following [16], using $V = 512$, $C = 8$, a batch size of 256, and 300 epochs per task. Given a horizon $H = 16$, we design *multi-scale* representations with scales of $1, 2, 3$ and 4 to capture *coarse-to-fine* information across the action sequence. We then train an autoregressive GPT-2 style, decoder-only transformer (see Sec. 3.2 of the main paper) , based on [26], using the same training settings as the benchmark, with a batch size of 256 for state-based experiments (4000 epochs) and a batch size of 64 for image-based experiments (3000 epochs). We use Cross-Entropy loss during training, which preserves the model's sampling capability. During inference, we typically select the token with the highest probability at each scale. However, to visualize multi-modal behavior in the Push-T task, we sample the top-k tokens at each scale (with k=3), allowing for diverse predictions with controlled randomness.

Multi-Task Simulation Experiment. To enable multi-task generalization, CARP augments the single-task formulation by introducing a learnable 3-dimensional task embedding for each task. These embeddings, retrieved based on task indices, are concatenated with the observation sequence $\boldsymbol{s}$ and act as additional conditional inputs to the policy. In our experiments involving 8 tasks, this corresponds to an 8×3 embedding matrix. We also use a moderately deeper decoder-only transformer in GPT-2 style. CARP is trained with a batch size of 512 for 200 epochs on an A100 GPU. Baseline models follow the same training settings as SDP [27]. This minimal modification enables CARP to adapt to multi-task learning seamlessly.

Real-World Experiment. For both baselines and CARP, the input consists of current visual observations from the wrist and scene cameras (resolution: 120×160), as well as proprioceptive data from the robotic arm. We execute 8 predicted actions out of a horizon of 16 predictions. We train the diffusion policy for 3000 epochs with a batch size of 64. For CARP, we use the same visual policy structure as in the simulation tasks, training the *multi-scale* action tokenizer for 300 epochs with a batch size of 256, and the *coarse-to-fine* transformer for 3000 epochs with a batch size of 64.

I. Consistent with Diffusion Policy

We adopt similar experimental settings with 1 or 2 observations, a prediction horizon of 16, and an executable action length of 8, following the standard setup used in Diffusion Policy (DP) [7]. It is important to note that our classical formulation introduces a minor discrepancy in the horizon definition compared to the implementation of DP. Specifically, in DP's experimental setting, the horizon H encompasses the past observed steps, meaning that the index of the current next predicted action is O, rather than 0. In contrast, as outlined in the formulation of Eq. (1) in the main paper, the horizon H does not include past observations, with the first prediction step corresponding to the next time step. While the rationale behind this design remains unclear due to limitations in the author's understanding, we retain the horizon definition introduced by Diffusion Policy (DP) [7] to ensure consistency in our experimental comparisons.

J. Ablation Study on the Number of Scales

To maintain consistency with Diffusion Policy, we set the action prediction horizon H to 16 across all tasks. Given $H = 16$, we adopt $K = 4$ for all experiments. To further investigate the impact of K, we conduct an ablation study by varying K from 1 to 6 on three representative tasks: Can, Square, and Kitchen (which requires executing four consecutive subtasks, thus we report success rate based on the final subtask, denoted as p4). All other experimental settings remain unchanged.

For each chosen number of scales K, the token map sizes at each scale level are defined as $1, \ldots, K$. Notably, when $K = 5$ and $K = 6$, the scales slightly exceed the default

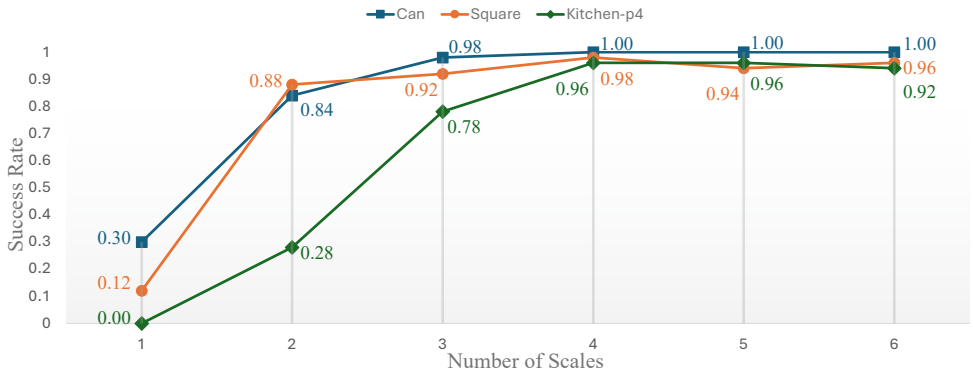

Figure 10. **Ablation Study on K.** We evaluate the performance of CARP across three tasks using six different scale configurations. Results indicate that when the number of scales exceeds 4, the model achieves optimal performance. Considering both model efficiency and performance, we set $K = 4$ in all experiments throughout the paper.

feature map size. To ensure fair evaluation, we appropriately expand the feature map size to accommodate these settings.

As shown in Fig. 10, using fewer scales leads to insufficient action tokenization, resulting in less precise predictions. When $K = 4$, the policy effectively meets task requirements. Further increasing K results in stable performance with negligible fluctuations, indicating that the policy has likely reached its peak performance for the given tasks, with additional scaling providing little to no further benefit.

K. Additional Real-World Experiment

We provide further visualization of the real-world experiments presented in the main paper. As shown in Fig. 11, CARP generates smooth and successful trajectories for the *Cup* and *Bowl* tasks, with temporal progression illustrated from left to right.

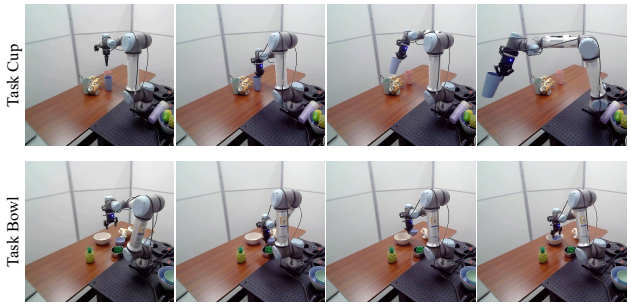

Figure 11. **Visualization of CAPR on Real-World Tasks.** CARP generates smooth and successful trajectories on the Cup and Bowl tasks, progressing from left to right.

Beyond the real-world evaluations on a UR5e robot arm, we further deploy CARP on a distinct robotic embodiment: a 7-DoF Franka Emika Panda arm. For this, we adopt the challenging *FurnitureBench* benchmark [9], which comprises long-horizon, contact-rich manipulation tasks (e.g., pick, place, insert, screw, and flip), with episodes spanning up to 1000 steps (700 steps for `One_Leg`). A corresponding standard simulator is also provided, as shown in Fig. 12.

We first evaluate CARP and Diffusion Policy (DP) in simulation on three tasks, followed by real-world deployment of the `One_Leg` task. All experiments are conducted in the state-based setting. To bridge the sim-to-real gap during real-world deployment, 6-DoF object poses are estimated using AprilTags provided by FurnitureBench [9], enabling consistent state-based policy execution.

For simulation, we use 200 trajectories per task from [3]. DP is trained using its official implementation [7] with 100 DDPM denoising steps. CARP follows the same state-based, single-task setting. We evaluate success rates using 1024 rollouts per task for statistical stability. As shown in Tab. 3, CARP achieves competitive performance while offering significantly lower inference time and fewer parameters.

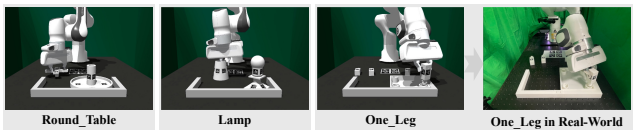

Figure 12. **FurnitureBench Tasks for Evaluation.** All three tasks are first evaluated in simulation. A corresponding real-world environment is then constructed to assess the performance in real-world.

Policy	One_Leg	Round_Table	Lamp	Inf.Time ↓	Params ↓
DP-C	39.62%	5.76%	3.91%	74.05s	66.06M
CARP	**43.75%**	**6.25%**	**4.30%**	**6.29**s	**2.54**M

Table 3. **Simulation Results on FurnitureBench (State-Based).** All policies are trained under identical single-task settings. Compared to DP-C [7] (with 100 DDPM denoising steps), CARP achieves comparable success rates while offering significantly lower inference time and parameter count.

We further evaluate CARP on the real-world `One_Leg` task using 40 expert demonstrations collected via a 3D SpaceMouse (left panel of Fig. 13). The task involves complex rotations and contact-rich interactions, as illustrated in the right panel of Fig. 13. CARP achieves higher success rates across key stages, especially in precision-critical steps such as *Insert*, demonstrating robust real-world performance.

L. Analysis on Fine-Grained Manipulation

Beyond standard pick-and-place tasks, which are relatively straightforward for robotic manipulation, tasks requiring fine-grained skills have garnered increasing attention. For example, Nut-Assembly and Threading from the MimicGen [19] benchmark demand precise action generation to ensure successful task completion, as illustrated in Fig. 14. In Nut-Assembly, the robot must place a nut onto a designated peg, which is slightly larger in size. Furthermore, the Threading task requires the robot to insert a needle into a small hole on a tripod, a significantly more challenging task due to the minimal margin for error. To evaluate CARP's fine-grained manipulation capability, we compare it with the state-of-the-art Sparse Diffusion Policy (SDP) [27] in the multi-task setting, and Diffusion Policy (DP) [7] in the single-task setting. We evaluate success rates alongside the mean and variance of the distance between the ideal insertion centers of the fixed structures (peg, tripod) and the centers of the tools (nut, needle) at the moment of first contact. A lower mean distance indicates higher action precision, while a smaller variance reflects the model's ability to consistently achieve accurate and stable manipulations.

As summarized in Tab. 4 and Tab. 5, our *coarse-to-fine* autoregressive prediction framework demonstrates strong performance in fine-grained tasks, achieving competitive results comparable to diffusion-based policies. Notably, CARP consistently achieves lower mean error and reduced variance across most tasks, regardless of whether in single-task or multi-task settings. Moreover, CARP achieves these results with an inference speed that is **10** times faster than current diffusion-based policies, highlighting its efficiency and effectiveness in fine-grained robotic manipulation.

M. Task Visualizations

In this section, we provide visualizations of the tasks used in our experiments. For the single-task experiment, the corresponding visualizations are presented in Fig. 15. For the multi-task experiment, visualizations are shown in Fig. 16. For the long-horizon, multi-stage Kitchen experiment, we provide visualizations in Fig. 17, along with the sequential execution process in Fig. 18. Finally, for real-world experiments, visualizations are included in Fig. 19.

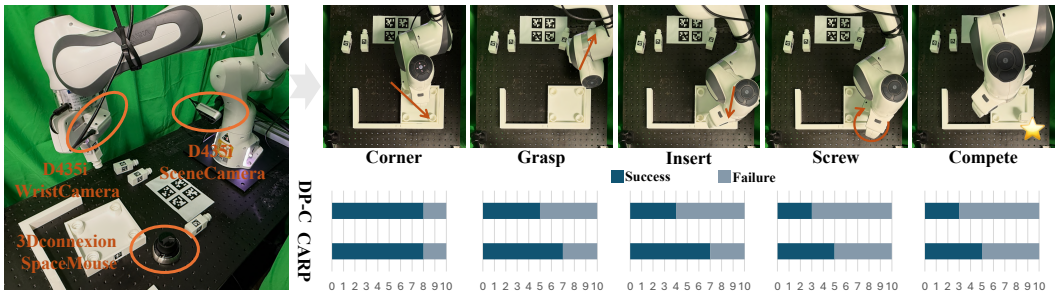

Figure 13. **Real-World Evaluation on the `One_Leg` Task.** The left panel shows the real-world setup, while the right panel illustrates the execution process and stage-wise results (left to right). Success rates at key stages are reported below. CARP produces smoother motions and outperforms baselines in precision-critical phases such as *Grasp* and *Insert*.

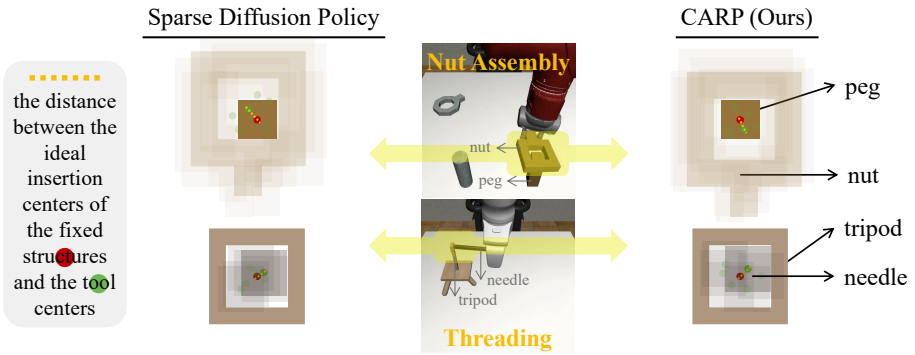

Figure 14. **Visualization of Fine-Grained Manipulation.** We evaluate the precision of generated actions by measuring the distance between the ideal and actual contact centers, represented by the dotted yellow line. Experiments are conducted on Nut-Assembly (top row) and Threading (bottom row). The visualization highlights that CARP achieves a comparable level of precision to diffusion-based policies.

Policy	Inference Speed ↑	*Nut Assembly*			*Threading*		
		Success ↑	Mean ↓	Variance ↓	Success ↑	Mean ↓	Variance ↓
SDP	8.2 hz	0.54	7.70	2.36	**0.70**	**5.20**	**1.25**
CARP	**118.5** hz	**0.66**	**7.30**	**1.68**	**0.70**	5.50	1.36

Table 4. **Fine-Grained Manipulation Study on Multi-Task Setting.** CARP demonstrates a high level of precision comparable to diffusion-based policies, as indicated by the similar mean and variance values. Additionally, CARP achieves a significant speed advantage, running over 10 times faster than diffusion-based approaches. This highlights CARP as a superior balance between performance and efficiency.

Policy	Inference Speed ↑	*Nut Assembly*			*Threading*		
		Success ↑	Mean ↓	Variance ↓	Success ↑	Mean ↓	Variance ↓
DP-C	10.13 hz	0.80	5.20	1.86	**0.88**	4.08	1.07
CARP	**119.05** hz	**0.82**	**5.12**	**1.28**	**0.88**	**3.92**	**0.94**

Table 5. **Fine-Grained Manipulation Study on Single-Task Setting.** To eliminate potential underfitting caused by multi-task training, we evaluate fine-grained tasks under single-task settings. CARP achieves success rates on par with DP-C, while offering lower variance and over 10× faster inference. These results highlight CARP's ability to maintain high precision and stability with greater inference efficiency.

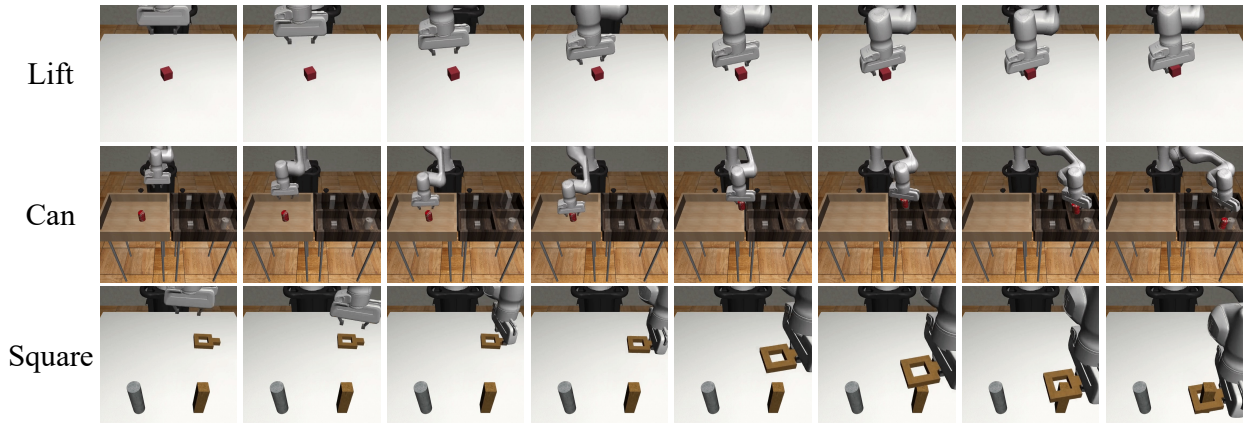

Figure 15. **Visualization of Tasks in Single-Task Experiment.**

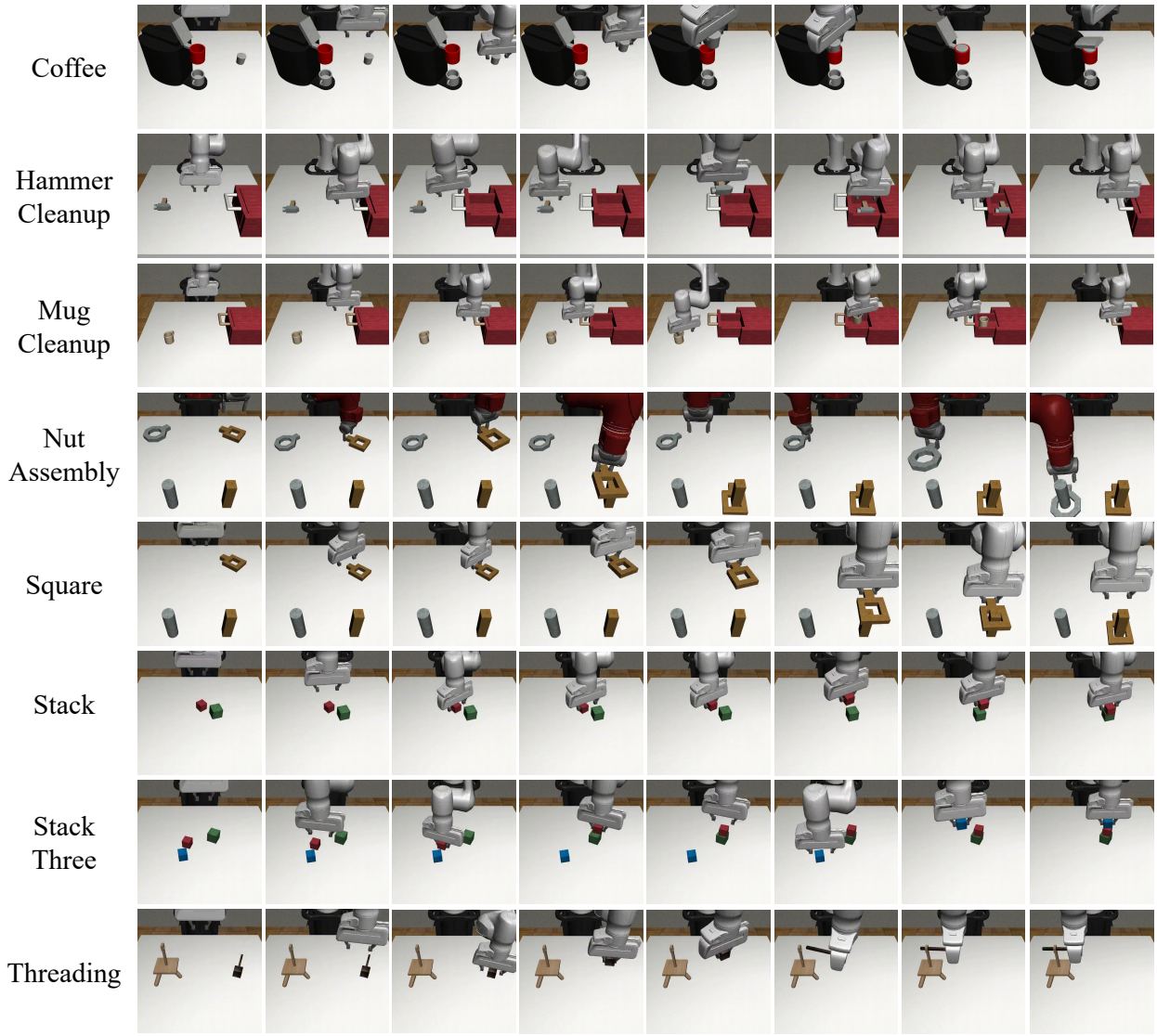

Figure 16. **Visualization of Tasks in Multi-Task Experiment.**

Figure 17. **Visualization of All Interaction Tasks in Kitchen Experiment.**

Figure 18. **Visualization of the Consecutive Execution in Kitchen Experiment.**

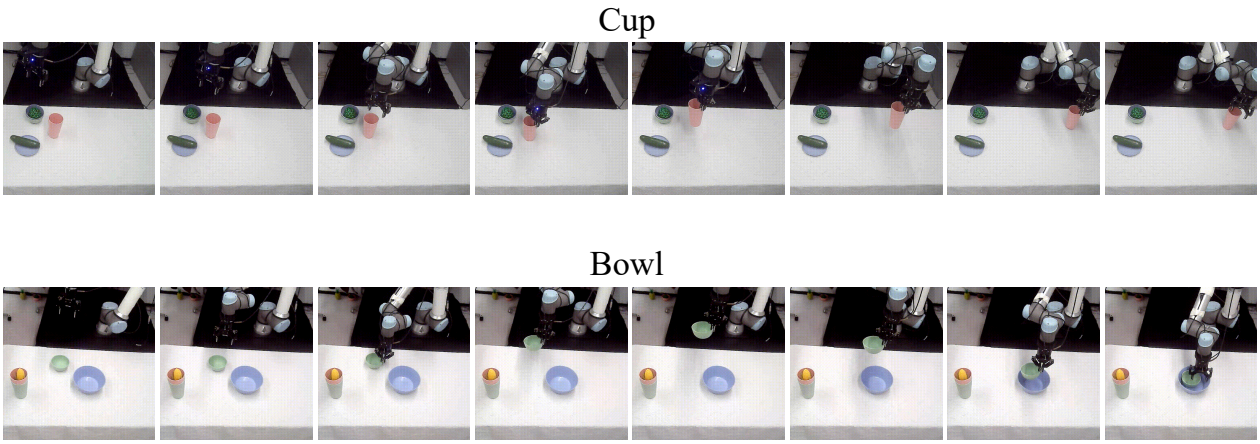

Figure 19. **Visualization of Tasks in Real-World Setup.**

References

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022. 1

[2] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022. 1

[3] Lars Ankile, Anthony Simeonov, Idan Shenfeld, Marcel Torne, and Pulkit Agrawal. From imitation to refinement–residual rl for precise assembly. *arXiv preprint arXiv:2407.16677*, 2024. 6

[4] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023. 1

[5] Tom B Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. 1

[6] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021. 1

[7] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023. 1, 3, 5, 6, 7

[8] Zichen Jeff Cui, Yibin Wang, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. From play to policy: Conditional behavior generation from uncurated robot data. *arXiv preprint arXiv:2210.10047*, 2022. 2

[9] Minho Heo, Youngwoon Lee, Doohyun Lee, and Joseph J Lim. Furniturebench: Reproducible real-world benchmark for long-horizon complex manipulation. *The International Journal of Robotics Research*, page 02783649241304789, 2023. 6

[10] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arxiv:2006.11239*, 2020. 1

[11] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34:1273–1286, 2021. 1

[12] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: General robot manipulation with multimodal prompts. *arXiv preprint arXiv:2210.03094*, 2(3):6, 2022. 2

[13] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. 1

[14] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024. 1

[15] Lucy Lai, Ann Zixiang Huang, and Samuel J Gershman. Action chunking as policy compression. *PsyArXiv*, 2022. 2

[16] Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11523–11532, 2022. 5

[17] Seungjae Lee, Yibin Wang, Haritheja Etukuru, H Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Behavior generation with latent actions. *arXiv preprint arXiv:2403.03181*, 2024. 2, 4

[18] Xinghang Li, Minghuan Liu, Hanbo Zhang, Cunjun Yu, Jie Xu, Hongtao Wu, Chilam Cheang, Ya Jing, Weinan Zhang, Huaping Liu, et al. Vision-language foundation models as effective robot imitators. *arXiv preprint arXiv:2311.01378*, 2023. 1

[19] Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. Mimicgen: A data generation system for scalable robot learning using human demonstrations. In *7th Annual Conference on Robot Learning*, 2023. 7

[20] Aaditya Prasad, Kevin Lin, Jimmy Wu, Linqi Zhou, and Jeannette Bohg. Consistency policy: Accelerated visuomotor policies via consistency distillation. *arXiv preprint arXiv:2405.07503*, 2024. 4

[21] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 1

[22] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 1

[23] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022. 2

[24] Nur Muhammad Mahi Shafiullah, Zichen Jeff Cui, Ariuntuya Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. In *Thirty-Sixth Conference on Neural Information Processing Systems*, 2022. 2, 4

[25] Garrett Thomas, Ching-An Cheng, Ricky Loynd, Felipe Vieira Frujeri, Vibhav Vineet, Mihai Jalobeanu, and Andrey Kolobov. Plex: Making the most of the available data for robotic manipulation pretraining. In *Conference on Robot Learning*, pages 2624–2641. PMLR, 2023. 2

[26] Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: Scalable im-

age generation via next-scale prediction. *arXiv preprint arXiv:2404.02905*, 2024. 5

[27] Yixiao Wang, Yifei Zhang, Mingxiao Huo, Ran Tian, Xiang Zhang, Yichen Xie, Chenfeng Xu, Pengliang Ji, Wei Zhan, Mingyu Ding, et al. Sparse diffusion policy: A sparse, reusable, and flexible policy for robot learning. *arXiv preprint arXiv:2407.01531*, 2024. 5, 7

[28] Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023. 2, 3