VOTE: Vision-Language-Action Optimization with Trajectory Ensemble Voting

Anonymous authors
Paper under double-blind review

Abstract

Recent large-scale Vision Language Action (VLA) models have shown superior performance in robotic manipulation tasks guided by natural language. However, current VLA models suffer from two drawbacks: (i) generation of massive tokens leading to high inference latency and increased training cost, and (ii) insufficient utilization of generated actions resulting in potential performance loss. To address these issues, we develop a training framework to finetune VLA models for generating significantly fewer action tokens with high parallelism, effectively reducing inference latency and training cost. Furthermore, we introduce an inference optimization technique with a novel voting-based ensemble strategy to combine current and previous action predictions, improving the utilization of generated actions and overall performance. Our results demonstrate that we achieve superior performance compared with state-of-the-art VLA models, achieving significantly higher success rates and 39× faster inference than OpenVLA with 46 Hz throughput on edge platforms, demonstrating practical deployability.

1 Introduction

Building general-purpose robotic policies capable of handling diverse tasks, embodiments, and real-world interactions has been a central challenge in robotics research. Recent studies Kim et al. (2024); Zhu et al. (2025); Qu et al. (2025); Li et al. (2024a) leverage Vision-Language-Action (VLA) models to address this problem, demonstrating excellent accuracy across a variety of robotic tasks. VLA models enable robots to perform complex tasks from natural language instructions, achieving outstanding performance on familiar objects and environments within the training distribution Brohan et al. (2023); O'Neill et al. (2024); Kim et al. (2025); Li et al. (2023). The VLA models are mainly developed based on the Vision Language Models (VLMs) Chen et al. (2023); Driess et al. (2023); Karamcheti et al. (2024); Beyer et al. (2024) through continuous training or finetuning on diverse robot data O'Neill et al. (2024); Fang et al. (2024). The success of this paradigm lies in leveraging the generalization capabilities of VLMs across diverse robotic manipulation tasks, alongside architectural designs that effectively integrate the VLM backbone with the robot action output head.

Recently works Li et al. (2024a); Zhu et al. (2025); Kim et al. (2025); Black et al. (2024) import the diffusion action head for optimization. Although diffusion demonstrates improved generalization capability, its scalability and practical applicability are limited by the high computational cost of both training and inference due to action diffusion. On the other hand, SpatialVLA Qu et al. (2025) attributes the poor generalization of traditional VLA models to insufficient access to high-level visual cues, such as 3D structure and depth information, arguing that the lack of such information limits generalization performance. To address this, SpatialVLA extends the VLA architecture by incorporating additional visual modules for 3D position encoding, yielding superior evaluation results. However, reliance on additional visual features introduces significant pre-processing overhead, and the increased number of visual tokens results in longer input sequences, further impacting inference speed. The high training cost across diverse robotic datasets, combined with the added latency of action sampling, limits the scalability and practical deployment of VLA models in real-world scenarios. This motivates us to investigate fast-acting efficient methods to reduce training and inference overhead.

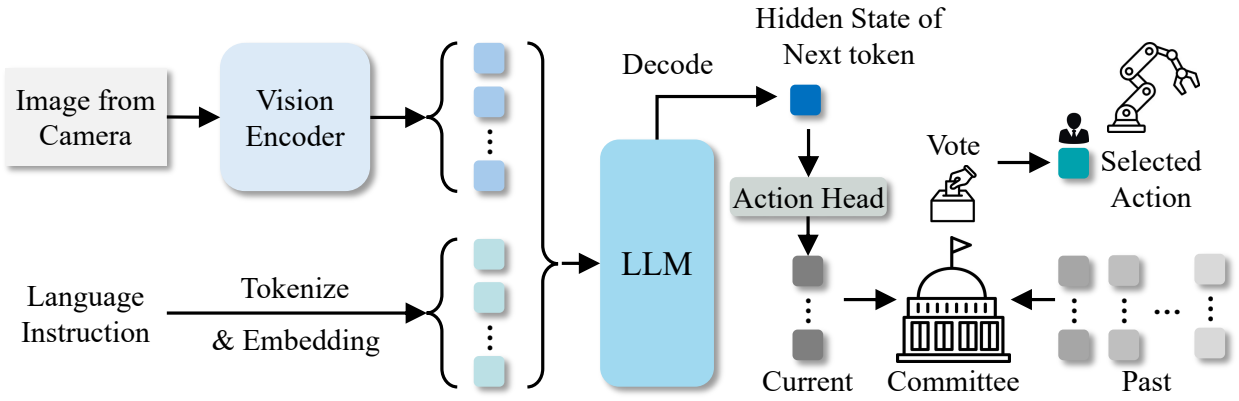

Figure 1: The whole VOTE pipeline, where we generate the next following n actions in parallel and adopt the ensemble voting strategy for the accurate current action prediction.

In this work, we propose **VOTE**, as shown in Figure 1, a lightweight VLA training method and a plug-and-play ensemble voting strategy for the optimized trajectory. To achieve higher throughput and faster inference in action sampling, we deliberately exclude additional visual modules and diffusion-based techniques, while preserving competitive performance. This approach greatly simplifies the pipeline for training new VLMs into VLAs, eliminating the need to consider complex action tokenizers.

Specifically, we introduce a special token `<ACT>` to represent entire action chunks. Thus, after finetuning, the model only needs to generate one single `<ACT>` token, instead of the original multiple tokens, significantly reducing the number of generated tokens and avoiding multiple sequential decoder passes with tokenization. Consequently, it enables faster inference and substantially reduces training cost, making rapid fine-tuning practical due to fewer required input-output tokens and simpler decoding processes. Meanwhile, we propose a novel action-ensemble technique, *ensemble voting*, to improve the model performance at test time. Specifically, we construct an action ensemble committee by incorporating actions predicted in previous steps, and determine the current action based on a voting mechanism weighted by the accumulated "tickets" from prior predictions. This sampling technique mitigates errors encountered by VLA models due to relying solely on the most recent inputs, reducing the likelihood of incorrect action predictions. Experimental results demonstrate that our method achieves state-of-the-art performance, while also delivering higher throughput and faster inference. We improves the average success rates of OpenVLA by over 20% across four LIBERO task suites, surpasses CogACT by 7% average success rate on SimplerEnv WidowX Robot, and accelerates action generation throughput by 39× on the edge device NVIDIA Jetson Orin.

Our contributions are summarized as follows:

1. We propose VOTE, including a training framework and an inference optimization technique. With the training framework, the VLA model only needs to generate one single special token instead of multiple tokens, thus effectively reducing the training and inference costs, while further improving the action success rate.
2. For the inference, we propose a novel action ensemble technique to construct a committee for action selection with voting, further improving the action success rate.
3. Experimental results show that our method achieves high success rate, with lower training costs and significantly improved inference speedups with higher throughput.

2 Related Work

Vision-Language-Action Models. Bridging the gap between seeing, understanding, and acting, Vision-Language-Action (VLA) models represent a significant leap in robotics and object manipulation. Although

VLMs excel at visual and language understanding, they are not inherently capable of generating actions for various robotic embodiments. More recently, several studies (Brohan et al., 2022; Zitkovich et al., 2023; Shentu et al., 2024; Li et al., 2024a; Qu et al., 2025; Black et al., 2024; Li et al., 2024b; 2025; Shi et al., 2025; Huang et al., 2025) present new ways of building general robot policies by fine-tuning pretrained VLMs on robot data, offering the ability to directly generate robot actions. RT-2-X Zitkovich et al. (2023) is a pioneering model that proposes a 55B VLA model pretrained on the Open X Embodiment (OXE) dataset O'Neill et al. (2024) with discretized actions. OpenVLA Kim et al. (2024) proposes to fine-tune the prismatic VLM Karamcheti et al. (2024) only on the OXE dataset O'Neill et al. (2024). CogACT Li et al. (2024a) employs a diffusion transformer-based action module to enhance generalization and adaptability in robotic tasks. Moreover, π_0 Black et al. (2024) finetunes PaliGemma VLM Beyer et al. (2024) and introduces a flow matching action head, which enables zero-shot and fine-tuned robotic control across diverse manipulation tasks. SpatialVLA Qu et al. (2025) introduces Ego3D Position Encoding to inject 3D information into the input observations of their VLA, representing spatial robot movement actions with Adaptive Action Grids. RoboVLM Li et al. (2024b) systematically transforms various VLMs into VLAs, exploring key design choices such as backbone selection, policy architecture, and cross-embodiment data integration.

Acceleration of VLA Models. The significant inference latency from intensive computations Zhan et al. (2024c;a); Shen et al. (2025b) limits VLA models from wide deployments on popular edge devices Zhan et al. (2024b); Shen et al. (2025a); Yang et al. (2023) and real-world robotic embodiments where real-time responsiveness is critical Zhao et al. (2024). Therefore, developing acceleration techniques is essential to advancing this field. Several innovative approaches have been developed for VLA models. DeeR-VLA Yue et al. (2024) introduces a dynamic early-exit framework for the backbone of VLA models, enabling the model to adaptively determine the computations required based on task complexity. VLA-Cache Xu et al. (2025) presents a token caching mechanism to adaptively identify and reuse unchanged visual tokens across sequential inputs to reduce redundant computations. TinyVLA Wen et al. (2024) and FAST Pertsch et al. (2025) focus on training smaller models from scratch, or applying new tokenization schemes to enhance the training time of VLA models. Recently, an optimized fine-tuning recipe was presented by OpenVLA-OFT Kim et al. (2025) to accelerate inference speed by integrating parallel decoding, action chunking, and continuous action representation.

3 Motivation

After investigating existing VLA models, we identify two key drawbacks: (i) generation of massive tokens or diffusion process leading to large inference latency and more training cost, and (ii) insufficient utilization of generated actions resulting in potential performance loss, which are detailed in the following.

3.1 Massive Computational Overhead

Typical VLA models need to predict multiple tokens corresponding to different action dimensions for each action, leading to high inference latency and training cost, whereas diffusion-based VLAs also introduce additional computational overhead from more training steps and multi-step denoising or integration during inference.

Large Inference Latency. We show the latency profile for SpatialVLA, OpenVLA and CogACT, in Figure 2. As observed, the primary computational overhead in current VLA models lies in the VLM backbone within the VLA architecture. The VLM decoding, which needs to generate large amounts of tokens, dominates the overall latency for action prediction with at least 50% occupation across three models. In particular, the diffusion part in CogACT incurs additional latency overhead. Meanwhile, SpatialVLA relies on multimodal high-level visual representations, such as 3D information, which needs to feed massive additional visual input tokens to the VLM with significantly increased latency.

Massive Training Cost. VLA models normally adopt finetuning with data from new tasks and embodiments to improve the performance in new environments Li et al. (2024a). Existing methods rely on large-scale pretraining data and additional downstream data (such as Fractal Brohan et al. (2022) and

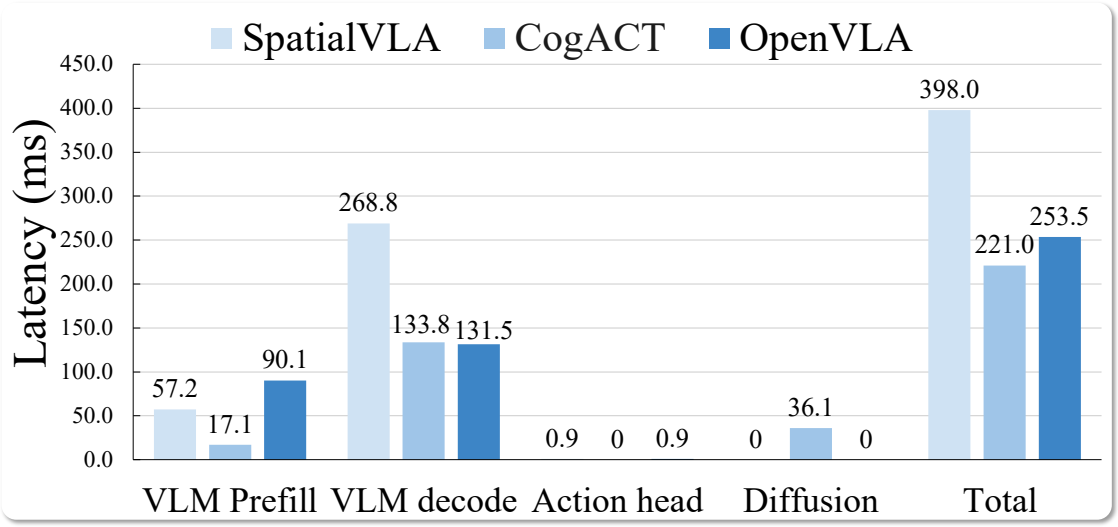

Figure 2: Latency for SpatialVLA, CogACT, and OpenVLA.

BridgeDataV2 Walke et al. (2023)) to adapt VLM backbones for robotic action prediction tasks. Moreover, during training, as we need to pad multiple empty action embeddings (corresponding to the number of output tokens for actions) as inputs, a large number of output tokens leads to padding massive input tokens, incurring significant additional training cost along with massive training data. OpenVLA-OFT Kim et al. (2025) shows that the diffusion action head converges more slowly and requires 1.67×–2× more gradient steps to converge compared to the MLP action head.

3.2 Insufficient Utilization of Generated Actions

Although VLA models generate large amounts of actions at high training and inference costs, we note that not all generated actions are utilized effectively. At each time step, the VLA model predicts a sequence of actions for the next multiple time steps. Thus, at each time step, the robot receives the action from the current inference, as well as the historical predictions for the current time step from previous model inferences.

Typically, the robots directly execute the action of the current time step from the current inference based on the current observation, discarding historical action predictions of previous time steps. This approach fails to fully utilize historical visual information and model predictions, leading to a less stable trajectory with potential performance degradation. To address this, prior works Zhao et al. (2023); Li et al. (2024a) propose to combine actions predicted for the current time step from both present and past inferences. However, these methods suffer from ineffective combination with meaningless outputs, too simple combination strategy, or heavy reliance on the current prediction which could be incorrect. Actions predicted at different timesteps can belong to different modes Chi et al. (2023), and simply aggregating them could result in an action that does not align with any modes. The utilization of generated actions from current and past inferences are insufficient, resulting in degraded performance.

4 Method

Motivated by the above drawbacks of the current VLA models, we develop a training framework to finetune VLA models for generating less action related tokens, thus addressing the first drawback with reduced inference latency and training cost. Furthermore, an inference optimization technique is proposed with a novel ensemble strategy to combine actions of current and previous predictions, thus addressing the second drawback with improved utilization of generations.

In this section, we first briefly provide the preliminaries of our model's architecture and problem statement. We next elaborate our innovative training method, detailing how the introduction of the special token <ACT> optimizes action prediction accuracy and computational efficiency. Then we introduce our novel vote-based adaptive action ensemble strategy, designed to further enhance the stability and robustness of action execution by dynamically selecting relevant actions based on cosine similarity.

4.1 Problem Statement

Our model generates an action based on the image $I \in \mathbb{R}^{W \times H \times 3}$ and the language instruction $\boldsymbol{l}$. At time step t, we utilize a model π to predict a temporal action sequence $(\boldsymbol{a}_t, \boldsymbol{a}_{t+1}, ..., \boldsymbol{a}_{t+N})$ for executing the desired task:

$$\boldsymbol{\pi} : (\boldsymbol{l}, \boldsymbol{I}_t) \rightarrow (\boldsymbol{a}_t, \boldsymbol{a}_{t+1}, ..., \boldsymbol{a}_{t+N}) \tag{1}$$

Here, $\boldsymbol{a}_t$ can describe various robot actions with different control modes and end-effectors. Following a strategy described in previous work Kim et al. (2024; 2025); Li et al. (2024a), we use 7 degrees of freedom (DoF) to express the end-effector pose of the robot arm:

$$\boldsymbol{a}_t = [\Delta x, \Delta y, \Delta z, \Delta\phi, \Delta\theta, \Delta\psi, g] \tag{2}$$

where $\Delta x, \Delta y, \Delta z$ are the relative translation offsets of the end effector, $\Delta\phi, \Delta\theta, \Delta\psi$ denote the rotation changes, and $g \in \{0, 1\}$ indicates the gripper's open/close state. This action space enables continuous control over robot arm motion and end-effector behavior.

4.2 Training Framework

Overview. During training, we introduce a special token <ACT> into the tokenizer of the LLM to explicitly signal the action prediction task. Specifically, we append this <ACT> token to the end of each language instruction sequence as the target token label. After the LLM performs a single forward pass to generates the single token, its hidden state from the final-layer is passed to the action head for transformation into continuous action values $\hat{\boldsymbol{a}}$. We specify the details of our training framework below.

Action Generation. Given a language instruction $\boldsymbol{l}$ and corresponding image I, the model generates multiple consecutive action predictions. First, the input data is processed by the VLA model to obtain hidden states:

$$\boldsymbol{h} = \text{VLA}(\boldsymbol{l}, I), \quad \text{where} \quad \boldsymbol{h} \in \mathbb{R}^{B \times \text{L} \times H}, \tag{3}$$

where B is the batch size, L is the sequence length, and H is the hidden dimension.

Next, the hidden states corresponding to the special action token <ACT> are extracted:

$$\boldsymbol{h}_{\texttt{<ACT>}} = \boldsymbol{h}[\text{mask}_{\texttt{<ACT>}}], \quad \text{where} \quad \boldsymbol{h}_{\texttt{<ACT>}} \in \mathbb{R}^{B \times 1 \times H}. \tag{4}$$

Note that here the model only needs to generate one single token <ACT> instead of multiple tokens for various multi-dimensional actions.

Then we need to convert the hidden state of <ACT> to actual action predictions. This is achieved with an action head. Specifically, the hidden state $\boldsymbol{h}_{\texttt{<ACT>}}$ is passed through an MLP Action Head to predict the action chunk (multiple consecutive actions). The Action Head first downsamples $\boldsymbol{h}_{\texttt{<ACT>}}$, then passes it through multiple bottleneck blocks. Each bottleneck block upsamples the dimension, then downsamples it again. The block architecture is shown below:

$$\begin{aligned} \boldsymbol{h}_{\text{norm}} &= \text{LayerNorm}(\boldsymbol{x}) \\ \boldsymbol{h}_{\text{mid}} &= \text{SiLU}\big(\boldsymbol{h}_{\text{norm}} \boldsymbol{W}_{\text{up}}\big) \\ \boldsymbol{h}_{\text{down}} &= \boldsymbol{h}_{\text{mid}} \boldsymbol{W}_{\text{down}} \\ \boldsymbol{h}_{\text{drop}} &= \text{Dropout}(\boldsymbol{h}_{\text{down}}) \\ \boldsymbol{out} &= \boldsymbol{x} + \boldsymbol{h}_{\text{drop}} \end{aligned} \tag{5}$$

This bottleneck design achieves better performance with fewer parameters compared with the isotropic architecture where all linear layers have the same dimension as $\boldsymbol{h}_{\texttt{<ACT>}}$.

The actions are obtained with an MLP action head for efficient parallel computing, rather than an action tokenizer with computation intensive decoding in traditional VLA models.

Training Objectives. Our training objective incorporates both token-level and action-level supervision.

We use $\boldsymbol{L}_{\text{action}}$ to represent the L1 loss between the predicted actions $\hat{\boldsymbol{a}}$ and the ground-truth actions $\boldsymbol{a}$ across all action dimensions.

$$\boldsymbol{L}_{\text{action}} = \boldsymbol{L}_{\boldsymbol{1}}(\hat{\boldsymbol{a}}, \boldsymbol{a}) = \frac{1}{BNA} \sum_{b=1}^{B} \sum_{n=1}^{N} \|\hat{\boldsymbol{a}}_{b,n} - \boldsymbol{a}_{b,n}\|_1, \tag{6}$$

Meanwhile, $\boldsymbol{L}_{\text{token}}$ is the cross-entropy loss calculated based on the prediction of the <ACT> token and all instruction tokens. These two losses are combined into a weighted total loss that balances semantic understanding from language modeling and accurate action generation:

$$\boldsymbol{L}_{\text{total}} = \lambda_{\text{token}} \boldsymbol{L}_{\text{token}} + \lambda_{\text{action}} \boldsymbol{L}_{\text{action}}, \tag{7}$$

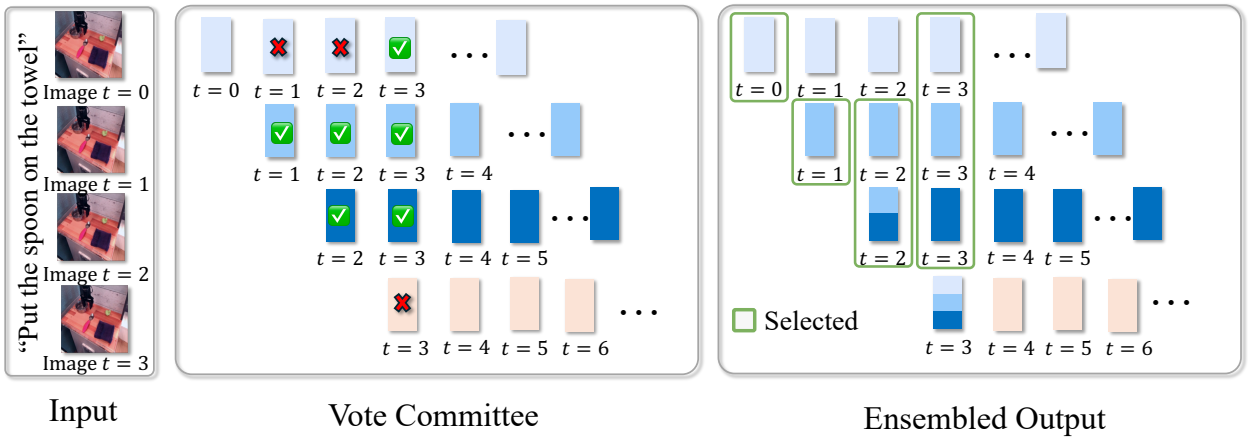

Figure 3: **Vote Action Ensemble.** Illustration of our action ensemble strategy with K = 3 (using the last 3 historical action predictions) as an example. Historical predictions and the current prediction form a voting committee to jointly determine the final action to execute. For example, when t = 3, more than half of the candidate actions differ from the current prediction, voting not-similar. Therefore, we discard the current prediction and instead compute the final ensembled action by averaging the previous 3 historical action predictions which vote not-similar.

Advantages for Training and Inference. Our method condenses the entire action chunk into a compact, high-level representation using a single <ACT> token. The hidden state of this token is passed through the action head to directly predict all action chunks. This significantly reduces the number of tokens required, leading to improved efficiency in both training and inference.

Specifically, for a chunk size of N time steps (i.e., N consecutive action predictions) with action dimensionality D, our method generates hidden states of one single token instead of ND hidden states corresponding to ND tokens for actions in the OpenVLA-OFT, which significantly reduces the number of generated tokens with non-marginal inference acceleration. Furthermore, when generating actions from hidden states, with our action head, we only require only a single forward pass instead of the ND sequential decoder forward passes in OpenVLA. In addition, adopting our method, due to the significantly reduced output token number, there is no need for padding ND empty action embeddings as input during OpenVLA-OFT training, thus reducing the training cost with fewer input/output training tokens and faster decoding.

Furthermore, instead of employing an action tokenizer to convert action tokens into actual actions as shown in OpenVLA or SpatialVLA, we directly utilize an action head to map the hidden state of the special token `<ACT>` to normalized continuous actions, enabling efficient parallel computation and eliminating the need for action tokenizer.

4.3 Ensemble Voting

During inference, the VLA model predicts a sequence of actions across multiple time steps. Typically, the robots execute these actions consecutively based on the current observation, discarding historical action predictions of previous time steps. However, this approach fails to fully utilize historical visual information and model predictions, leading to a less stable trajectory with potential performance degradation.

To fully utilize the generated actions, we propose a voting-based adaptive ensemble strategy for action aggregation, which selects the more frequent prediction (with a higher chance to be correct) from a list of action predictions from adjacent time steps. Specifically, given the current observation $\boldsymbol{o}_t$, let $(\boldsymbol{a}_t|\boldsymbol{o}_t)$ denote the predicted action at current time step t. Note that one inference can generate multiple consecutive actions, i.e., $(\boldsymbol{a}_t|\boldsymbol{o}_t)$, $(\boldsymbol{a}_{t+1}|\boldsymbol{o}_t)$, $(\boldsymbol{a}_{t+2}|\boldsymbol{o}_t)$, and so on. At the current time step t, the past action predictions from previous time steps are also available, represented as: $\boldsymbol{H} = \{(\boldsymbol{a}_t|\boldsymbol{o}_{t-K}), ..., (\boldsymbol{a}_t|\boldsymbol{o}_t)\}$.

We first compute the similarity between each action in $\boldsymbol{H}$ and the current/newest prediction $(\boldsymbol{a}_t|\boldsymbol{o}_t)$. Based on all these similarities, the action set $\boldsymbol{H}$ is split into two subsets, $\boldsymbol{M}$ for higher similarity and $\boldsymbol{N}$ for lower similarity. Following the common voting rule, we select the set with more votes and compute the average of all actions in the selected subset as the final action for the time step t. The ensemble action $\tilde{\boldsymbol{a}}_t$ at time step t is computed by the following:

$$\hat{\boldsymbol{a}}_t = \begin{cases} \dfrac{1}{|\boldsymbol{M}|} \displaystyle\sum_{\boldsymbol{x} \in \boldsymbol{M}} \boldsymbol{x}, & \text{if } |\boldsymbol{M}| > |\boldsymbol{N}|, \\ \dfrac{1}{|\boldsymbol{N}|} \displaystyle\sum_{\boldsymbol{x} \in \boldsymbol{N}} \boldsymbol{x}, & \text{otherwise,} \end{cases} \tag{8}$$

$$\boldsymbol{M} = \{(\boldsymbol{a}_t|\boldsymbol{o}_{t-k}) \mid \langle \boldsymbol{a}_t|\boldsymbol{o}_t, \boldsymbol{a}_t|\boldsymbol{o}_{t-k} \rangle > \tau,\ k \in \{0, \ldots, K\}\}, \tag{9}$$

$$\boldsymbol{N} = \{(\boldsymbol{a}_t|\boldsymbol{o}_{t-k}) \mid \langle \boldsymbol{a}_t|\boldsymbol{o}_t, \boldsymbol{a}_t|\boldsymbol{o}_{t-k} \rangle \leq \tau,\ k \in \{0, \ldots, K\}\}, \tag{10}$$

where $\langle \cdot, \cdot \rangle$ denotes cosine similarity, τ is a threshold empirically set to 0.5, and $|\cdot|$ is the element number in a set.

Advantages. (i) Unlike straightforward action chunking, where actions are executed consecutively, or static weighted aggregation methods in Zhao et al. (2023), our method selects the actions with more votes, thus more likely to be correct. (ii) Although the naive average method to compute the average of all actions in $\boldsymbol{H}$ is straightforward, it may take the incorrect predictions into considerations with performance degradations. Different from the average method or adaptive method Li et al. (2024a), our voting ensemble effectively filters out unreasonable or inconsistent mode predictions, thereby enhancing the reliability and robustness of the final aggregated action. (iii) Our method pays more attention or gives more credit to the current/newest action prediction, since all similarities are computed with reference to the current action with one definite vote for high similarity. This is reasonable as the current observation provides the most critical real-time information. (iv) In the case that the current prediction $(\boldsymbol{a}_t|\boldsymbol{o}_t)$ is incorrect, if more previous predicted actions are different/unlike the current prediction, i.e., do not vote for the current prediction, our method will disregard the current prediction and ensemble the previous predicted actions which vote NO with less similarity to the incorrect current prediction and higher likelihood to be correct. (v) The experiments demonstrate that our method can outperform the straightforward average or static-weighted aggregation method.

Method	Put Spoon(%)	Put Carrot(%)	Stack Block(%)	Put Eggplant(%)	Average(%)	Latency(ms)↓	Speed up↑
RT-1-X	0.0	4.2	0.0	0.0	1.1	–	–
Octo-Base	12.5	8.3	0.0	43.1	16.0	–	–
Octo-Small	47.2	9.7	4.2	56.9	30.0	–	–
OpenVLA	0.0	0.0	0.0	4.1	1.0	240	1.0
RoboVLM	29.2	25.0	12.5	58.3	31.3	–	–
π_0	29.1	0	16.6	62.5	27.1	–	–
π_0-FAST	29.1	21.9	10.8	66.6	32.1	470	0.5
SpatialVLA	16.7	25.0	29.2	100.0	42.7	400	0.6
CogACT	71.7	50.8	15.0	67.5	51.3	220	1.1
Ours	58.3	29.2	**50.0**	**95.8**	**58.3**	**78**	**3.1**

Table 1: Evaluation results on the SimplerEnv WidowX robot setting. *Stack Block* refers to "Stack Green Block on Yellow Block" task, *Put Eggplant* to "Put Eggplant in Yellow Basket" task, *Put Carrot* to "Put Carrot on Plate" task, and *Put Spoon* to "Put Spoon on Towel" task. The zero-shot and fine-tuning results denote the performance of models pretrained on the OXE dataset and models fine-tuned on the BridgeData V2, respectively. Latency is tested on A6000 GPU.

5 Experimental Results

5.1 Experimental Setup and Baselines

We evaluate our model on the LIBERO (Liu et al., 2023) and SimplerEnv (Li et al., 2024c) simulation benchmarks, which comprise a diverse set of robotic manipulation tasks in simulated environments. All simulated evaluations were conducted on NVIDIA RTX A6000 and H100 GPUs. We fine-tune on OpenVLA model using AdamW with a learning rate of 1×10^{-4}. Fine-tuning employs Low-Rank Adaptation (LoRA) (Hu et al., 2022) with rank $r = 32$ and $\alpha = 16$. For LIBERO, we train on 2 H100 GPUs with a global batch size of 40.

For the token loss, we require the model to correctly predict the `<ACT>` token. If the token loss weight is too small, the model struggles to recognize and attend to `<ACT>`. If the weight is too large, the loss becomes overly dominated by token prediction, undermining the accuracy and convergence of action prediction. We experimented with action loss weights from 0.5 to 0.99, and observed the best performance when setting the action loss weight to 0.99 and the token loss weight to 0.01.

For the learning rate, we performed a grid search over learning rates in the range from 2e-5 to 5e-4 and observed that 1e-4 yielded the best overall performance. Learning rates smaller than 1e-4 led to slow convergence and suboptimal performance, while a higher rate caused unstable gradients and poor training stability.

Training Details for SimplerEnv. We train on 4 H100 GPUs with a global batch size of 80. The action chunk size N and ensemble horizon $K + 1$ (including the current prediction) are both set to 8. We finetune on same datasets as Qu et al. (2025). For **WidowX robot** simulations in the SimplerEnv, we fine-tune 60 K steps on the BridgeDataV2 dataset Walke et al. (2023). For **Google robot** simulations in the SimplerEnv, we fine-tune 70 K steps on Fractal dataset.

5.2 Evaluation Results on SimplerEnv

To evaluate the robustness of our model under diverse environmental variations, we use the SimplerEnv simulation benchmark Li et al. (2024c), which measures visual matching and variant aggregation metrics. SimplerEnv offers diverse manipulation scenarios with changes in lighting, color, texture, and camera pose. It is designed to bridge the real to sim control and visual gap by faithfully replicating real world conditions for robots like the Google robot and the WidowX robot. Extensive testing of various VLA models has shown a strong correlation between performance in SIMPLER and real-world outcomes.Li et al. (2024c).

Models	Spatial SR (%)	Object SR (%)	Goal SR (%)	Long SR (%)	Average SR (%)
OpenVLA	84.7	88.4	79.2	53.7	76.5
Diffusion Policy	78.3	92.5	68.3	50.5	72.4
Octo	78.9	85.7	84.6	51.1	75.1
TraceVLA	84.6	85.2	75.1	54.1	74.8
SpatialVLA	88.2	89.9	78.6	55.5	78.1
π_0-FAST	96.4	96.8	88.6	60.2	85.5
π_0	96.8	98.8	95.8	85.2	94.2
OpenVLA-OFT	96.2	98.3	96.2	90.7	95.3
Ours	**98.8**	**99.8**	**97.6**	**95.6**	**98.0**

Table 2: Success rates (SR) across LIBERO benchmark task suites. Chunk size is 8. Ours achieves the highest SR.

Table 1 summarizes the results across different manipulation policies on the WidowX setup within SimplerEnv. Each task repeats 24 trails. Our model surpasses state-of-the-art methods such as CogACT (Li et al., 2024a) and SpatialVLA, with an average success rate of 58.3%. We report results for Google Robot within the SimplerEnv in Appendix.

Table 1 also demonstrates the latency and speedup in SimplerEnv. Our method achieves more than 3× speedup over OpenVLA.

Models	Spatial SR (%)	Object SR (%)	Goal SR (%)	Long SR (%)	Average SR (%)
Isotropic (Chunk=8)	98.0	99.5	96.0	94.0	96.9
Isotropic (Chunk=16)	96.0	98.5	94.0	91.0	94.9
Bottleneck (Chunk=8)	**98.8**	**99.8**	**97.6**	**95.6**	**98.0**
Bottleneck (Chunk=16)	97.6	97.8	96.8	93.8	96.5

Table 3: LIBERO performance comparison across different architectures and chunk sizes.

Model	Chunk Size	Platform	Latency (ms) ↓	Peak VRAM (GB) ↓	Throughput (Hz) ↑	Speed up↑
OpenVLA	1	A6000	240	14.35	4.2	1.0
SpatialVLA	4	A6000	400	7.82	10.1	2.4
OpenVLA-OFT	8	A6000	88	19.20	90.9	21.6
CogACT	16	A6000	220	29.33	72.4	17.7
Ours	8	A6000	78	14.40	102.6	24.4
Ours	16	A6000	78	14.40	**205.2**	**48.8**
OpenVLA	1	Orin	836	14.35	1.2	1.0
SpatialVLA	4	Orin	1949	7.82	2.1	1.7
OpenVLA-OFT	8	Orin	342	19.20	23.4	19.6
CogACT	16	Orin	–	OOM	–	–
Ours	8	Orin	346	14.40	23.1	19.3
Ours	16	Orin	346	14.40	46.2	38.6

Table 4: **Cross-Platform Inference Evaluation**. Peak VRAM represents the maximum GPU memory used during inference. Speedup is reported relative to OpenVLA as the baseline.

5.3 Evaluation Results on LIBERO

The evaluation results on LIBERO are shown in Table 2. As observed, our method with a chunk size 8 performs best in most of sub-tasks for the LIBERO benchmark. The results demonstrate that our method improves the success rate.

We analyze the impact of chunk size and model architecture on overall performance. Table 3 compares models with chunk sizes of 8 and 16 under both isotropic and bottleneck architectures. We observe that the bottleneck architecture consistently outperforms the isotropic one across both chunk sizes. This performance gap can be attributed to the inductive bias introduced by the bottleneck architecture. The bottleneck architecture compels the model to learn more compact and meaningful intermediate representations. In the downsampling phase, the model is forced to extract the most essential feature information; during upsampling, it reconstructs outputs based on these core features. This architectural constraint effectively guides the model toward learning more informative feature representations.

Our model maintains superior performance relative to baseline methods when predicting 16 subsequent actions from a single observation, exhibiting only a modest 1.5% reduction in average success rate compared to a chunk size of 8.

5.4 Cross-Platform Inference Evaluation

To investigate the efficiency of **VOTE**, we measured the average latency (i.e., the time to generate an action chunk) and throughput (i.e., the number of actions generated per second) by querying each model 100 times on distinct platforms. Each query processes a 224×224 image and a sample language instruction ("*What action should the robot take to pick the cup?*"). Orin specifications can be found in Appendix.

We first test the inference latency on the A6000 GPU. As shown in Table 4, **VOTE** achieves a throughput of approximately 20× of SpatialVLA Qu et al. (2025), despite the larger LLM used in our model (our LLaMA2-7B versus SpatialVLA's PaliGemma-3B). With chunk 16, **VOTE** can deliver up to 48.8× speed up compared to OpenVLA, outperforming other baselines. The difference in speedups compared to Table 1 arises because of different evaluation setting. In Table 1, we use the same setting as CogACT and SpatialVLA to predict one action chunk at each timestep without finishing executing all actions in this chunk. But in Table 4, we use the same setting as OpenVLA-OFT to predict the next action chunk only after all actions in the chunk have been executed.

Meanwhile, modern edge-computing platforms, such as the NVIDIA AGX Orin Leela (2022), are preferred for real-time robotic control, enabling real-time robot inference and low latency. However, these platforms struggle when faced with the heavy demands of VLA models due to limited and heterogeneous computing resources. To assess performance on the edge platform, we compare our proposed approach with existing methods on OpenVLA Kim et al. (2024), SpatialVLA Qu et al. (2025), CogACT Li et al. (2024a), and OpenVLA-OFT Kim et al. (2025). As shown in Table 4, **VOTE** (with a chunk size of 16) achieves 46 Hz throughput and a 38.6× speedup over OpenVLA, while imposing negligible memory overhead (0.7%) compared with 33.8% more memory cost for OpenVLA-OFT, whereas CogACT fails to execute due to Out-of-Memory (OOM). These results highlight our superior latency and throughput, making our approach well-suited for edge deployment.

Strategy	GR VM SR (%)	GA VG SR (%)	WR SR (%)	Average SR (%)
None	60.9	56.3	24.0	47.1
Temporal	70.5	60.3	30.2	53.7
Adaptive	71.9	60.3	49.0	60.4
Average	72.1	59.6	53.2	61.6
Vote	**74.9**	60.2	**58.3**	**64.5**

Table 5: Comparison of our proposed action ensemble strategy, Vote Ensemble, with other strategies. The Average strategy simply averages all historical predictions.

Models	Params	GR VM SR (%)	GR VA SR (%)	WR SR (%)	Average SR (%)
Isotropic	50M	68.6	63.6	56.2	62.8
Bottleneck	37M	74.9	60.2	58.3	64.5

Table 6: Comparison of success rates (SR) between bottleneck and isotropic architectures.

5.5 Ablation Study

We conduct ablation studies using the SIMPLER evaluation on both the Google Robot (GR) and the WidowX Robot (WR). We use the following abbreviations: VM for the SIMPLER Visual Matching setting, and VA for the SIMPLER Visual Aggregation setting.

We present an action ensemble strategy, termed Vote Ensemble, as formulated in Eq. equation 8. We ablate the effects of Vote Ensemble in Table 5. The temporal strategy is introduced by Zhao et al. (2023), while the adaptive strategy is proposed by Li et al. (2024a). Our proposed Vote Ensemble outperforms others, and we attribute this to its integration of similarity weighting between current and historical predictions, as well as the voting based majority ensemble.

Table 6 presents a comparison between isotropic and bottleneck design. Our bottleneck design achieves better performance with fewer parameters compared with the isotropic architecture where all linear layers have the same dimension as $\boldsymbol{h}_{\texttt{<ACT>}}$.

Our method is robust to the choice of the hyperparameter τ. We achieves the best results when τ is 0.5, and outperforms other strategies across a broader range from 0.0 to 0.8. As shown in Figure 4, our method demonstrates robustness to hyperparameter τ without requiring careful tuning. It achieves optimal performance of 58.3% when $\tau = 0.5$, and outperforms other strategies across a broader τ range from 0.0 to 0.8. This is because the ensemble approach inherently provides a degree of fault tolerance and stability, as it considers multiple predictions rather than relying on a potentially incorrect prediction.

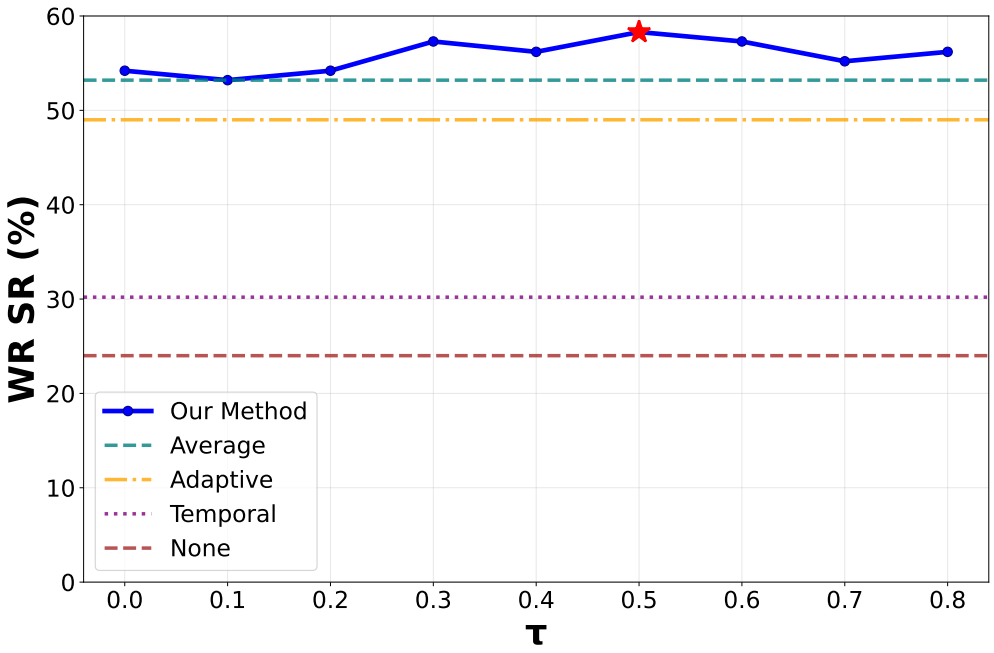

Figure 4: Ablation on hyperparameter τ. WR denotes WidowX Robot in SimplerEnv setting. τ is a hyperparameter specific to Our Method and that the baselines are shown for comparison and do not depend on τ

6 Conclusion

We have presented a lightweight VLA framework that enhances efficiency by predicting actions in a hidden latent space. Our approach leverages a action-tokenizer-free training methodology that simultaneously predicts multiple actions with bottleneck action head, significantly reducing computational requirements during both training and inference. Furthermore, we propose a straightforward yet effective action ensemble algorithm that optimizes action sampling. Extensive experimental results confirm that our model achieves superior inference speedups, while exhibiting exceptional generative performance.

References

Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, et al. Paligemma: A versatile 3b vlm for transfer. *arXiv preprint arXiv:2407.07726*, 2024.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023. URL `https://arxiv.org/abs/2307.15818`.

Xi Chen, Josip Djolonga, Piotr Padlewski, Basil Mustafa, Soravit Changpinyo, Jialin Wu, Carlos Riquelme Ruiz, Sebastian Goodman, Xiao Wang, Yi Tay, et al. Pali-x: On scaling up a multilingual vision and language model. *arXiv preprint arXiv:2305.18565*, 2023.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, pp. 02783649241273668, 2023.

Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

Hao-Shu Fang, Hongjie Fang, Zhenyu Tang, Jirong Liu, Chenxi Wang, Junbo Wang, Haoyi Zhu, and Cewu Lu. Rh20t: A comprehensive robotic dataset for learning diverse skills in one-shot. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 653–660. IEEE, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Huang Huang, Fangchen Liu, Letian Fu, Tingfan Wu, Mustafa Mukadam, Jitendra Malik, Ken Goldberg, and Pieter Abbeel. Otter: A vision-language-action model with text-aware visual feature extraction. *arXiv preprint arXiv:2503.03734*, 2025.

Siddharth Karamcheti, Suraj Nair, Ashwin Balakrishna, Percy Liang, Thomas Kollar, and Dorsa Sadigh. Prismatic vlms: Investigating the design space of visually-conditioned language models. In *Forty-first International Conference on Machine Learning*, 2024.

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.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

S Karumbunathan Leela. NVIDIA Jetson AGX Orin Series Technical Brief: a giant leap forward for robotics and edge AI applications, 2022. URL `https://www.nvidia.com/content/dam/en-zz/Solutions/gtcf21/jetson-orin/nvidia-jetson-agx-orin-technical-brief.pdf`.

Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, et al. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024a.

Xinghang Li, Minghuan Liu, Hanbo Zhang, Cunjun Yu, Jie Xu, Hongtao Wu, Chilam Cheang, Ya Jing, Weinan Zhang, Huaping Liu, Hang Li, and Tao Kong. Vision-language foundation models as effective robot imitators. *arXiv preprint arXiv:2311.01378*, 2023.

Xinghang Li, Peiyan Li, Minghuan Liu, Dong Wang, Jirong Liu, Bingyi Kang, Xiao Ma, Tao Kong, Hanbo Zhang, and Huaping Liu. Towards generalist robot policies: What matters in building vision-language-action models. *arXiv preprint arXiv:2412.14058*, 2024b.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024c.

Yi Li, Yuquan Deng, Jesse Zhang, Joel Jang, Marius Memmel, Raymond Yu, Caelan Reed Garrett, Fabio Ramos, Dieter Fox, Anqi Li, et al. Hamster: Hierarchical action models for open-world robot manipulation. *arXiv preprint arXiv:2502.05485*, 2025.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6892–6903. IEEE, 2024.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Yan Ding, Zhigang Wang, JiaYuan Gu, Bin Zhao, Dong Wang, et al. Spatialvla: Exploring spatial representations for visual-language-action model. *arXiv preprint arXiv:2501.15830*, 2025.

Xuan Shen, Weize Ma, Jing Liu, et al. Quartdepth: Post-training quantization for real-time depth estimation on the edge. In *CVPR*, 2025a.

Xuan Shen, Hangyu Zheng, Yifan Gong, Zhenglun Kong, Changdi Yang, Zheng Zhan, Yushu Wu, Xue Lin, Yanzhi Wang, Pu Zhao, and Wei Niu. Sparse learning for state space models on mobile. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL `https://openreview.net/forum?id=t8KLjiFNwn`.

Yide Shentu, Philipp Wu, Aravind Rajeswaran, and Pieter Abbeel. From llms to actions: Latent codes as bridges in hierarchical robot control. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8539–8546. IEEE, 2024.

Lucy Xiaoyang Shi, Brian Ichter, Michael Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, et al. Hi robot: Open-ended instruction following with hierarchical vision-language-action models. *arXiv preprint arXiv:2502.19417*, 2025.

Homer Rich Walke, Kevin Black, Tony Z Zhao, Quan Vuong, Chongyi Zheng, Philippe Hansen-Estruch, Andre Wang He, Vivek Myers, Moo Jin Kim, Max Du, et al. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning*, pp. 1723–1736. PMLR, 2023.

Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, Yaxin Peng, Feifei Feng, and Jian Tang. Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation, 2024. URL https://arxiv.org/abs/2409.12514.

Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. Vla-cache: Towards efficient vision-language-action model via adaptive token caching in robotic manipulation. *arXiv preprint arXiv:2502.02175*, 2025.

Changdi Yang, Pu Zhao, Yanyu Li, et al. Pruning parameterization with bi-level optimization for efficient semantic segmentation on the edge. In *CVPR*, 2023.

Yang Yue, Yulin Wang, Bingyi Kang, Yizeng Han, Shenzhi Wang, Shiji Song, Jiashi Feng, and Gao Huang. Deer-vla: Dynamic inference of multimodal large language models for efficient robot execution. *Advances in Neural Information Processing Systems*, 37:56619–56643, 2024.

Zheng Zhan, Zhenglun Kong, Yifan Gong, Yushu Wu, Zichong Meng, Hangyu Zheng, Xuan Shen, Stratis Ioannidis, Wei Niu, Pu Zhao, and Yanzhi Wang. Exploring token pruning in vision state space models. In *NeurIPS*, 2024a.

Zheng Zhan, Yushu Wu, Yifan Gong, et al. Fast and memory-efficient video diffusion using streamlined inference. In *NeurIPS*, 2024b. URL https://openreview.net/forum?id=iNvXYQrkpi.

Zheng Zhan, Yushu Wu, Zhenglun Kong, Changdi Yang, Yifan Gong, Xuan Shen, Xue Lin, Pu Zhao, and Yanzhi Wang. Rethinking token reduction for state space models. In *EMNLP*, pp. 1686–1697, Miami, Florida, USA, nov 2024c. ACL. URL https://aclanthology.org/2024.emnlp-main.100.

Pu Zhao, Fei Sun, Xuan Shen, Pinrui Yu, Zhenglun Kong, Yanzhi Wang, and Xue Lin. Pruning foundation models for high accuracy without retraining. In *Findings of EMNLP 2024*, pp. 9681–9694. ACL, November 2024. doi: 10.18653/v1/2024.findings-emnlp.566. URL https://aclanthology.org/2024.findings-emnlp.566.

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.

Minjie Zhu, Yichen Zhu, Jinming Li, Zhongyi Zhou, Junjie Wen, Xiaoyu Liu, Chaomin Shen, Yaxin Peng, and Feifei Feng. Objectvla: End-to-end open-world object manipulation without demonstration. *arXiv preprint arXiv:2502.19250*, 2025.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pp. 2165–2183. PMLR, 2023.

Appendix

A Evaluation Details

A.1 Baselines

We compare our model with various manipulation policies:

- **RT-1/RT-1-X/RT-2-X/Octo/OpenVLA/HPT/ TraceVLA/RoboVLM**: trained on mixtures of the Open X-Embodiment (OXE) dataset O'Neill et al. (2024).
- π_0 and π_0**-FAST**: trained on 90.9% proprietary data and 9.1% open datasets, including BridgeDataV2 Walke et al. (2023), DROID, and OXE.
- **SpatialVLA**: pre-trained on a mixture of OXE and RH20T Fang et al. (2024), and fine-tuned on BridgeDataV2/Fractal for SimplerEnv benchmark.
- **CogACT**: fine-tuned from OpenVLA checkpoint using the OXE dataset.

A.2 LIBERO Evaluation Detail

The LIBERO benchmark has 4 task suites, which evaluate the model's understanding of spatial relationships (**LIBERO-Spatial**), object types (**LIBERO-Object**), task-oriented behaviors (**LIBERO-Goal**), and its ability to generalize to long-horizon tasks with diverse objects, layouts, and goals (**LIBERO-Long**). In our experiment, we conduct evaluations across four task suites, each containing 10 tasks. Each task is repeated 50 times, resulting in a total of 500 trials per suite. The random seed for the evaluation is set to 7.

A.3 SimplerEnv Evaluation Detail

SimplerEnv Simulation Environment offers two evaluation settings: Visual Matching, which closely replicates real-world tasks by minimizing discrepancies between the simulated and real environments, and Variant Aggregations, which introduces variations to Visual Matching by modifying elements such as background, lighting, distractors, and table texture.

Models are evaluated every 5 K steps because action loss alone is not fully indicative of performance. We report results for Google Robot within the SimplerEnv in Table 8. SimplerEnv results are evaluated on NVIDIA RTX A6000 GPUs (48 GB VRAM each) with 256 GB system RAM. SimplerEnv is built on ManiSkill2 as its base simulation environment. Maniskill2 SAPIEN random seed is 2022.

B Training Details

Training runs on NVIDIA H100 NVL GPUs (94 GB VRAM each) with 756 GB RAM. We use a shuffle buffer size of 100K, random seed of 7, and dropout rate of 0.1. For libero benchmark, training was continued until the mean L1 loss between predicted and ground-truth normalized actions less than 0.03.

We compare our training effort with two baselines: SpatialVLA and OpenVLA-OFT, which use the same training dataset and report their training steps. As shown in Table 7, our method consistently outperforms the SpatialVLA across four LIBERO tasks while demonstrating superior training efficiency. For instance, on the LIBERO-Long task, our approach achieves 95.6 % success rate (SR), 40.1 % improvement over SpatialVLA. This significant performance gain is achieved with less training cost. While the SpatialVLA baseline is fine-tuned for 200 epochs, our method converges much more rapidly: 5 epochs for LIBERO-Object, 35 epochs for LIBERO-Goal and LIBERO-Spatial, and 50 epochs for the LIBERO-Long task. On average, our approach requires only 15.6 % of the training efforts. The Libero training loss figure is shown in Figure 5.

Task	SpatialVLA SR(%)	Ours SR (%)	Training Effort (%) ↓
Spatial	88.2	**98.8**	17.5
Object	89.9	**99.8**	2.5
Goal	78.6	**97.6**	17.5
Long	55.5	**95.6**	25.0
Average	78.1	**98.0**	15.6

Table 7: Comparison of LIBERO task success rates (SR) between SpatialVLA and our method, along with training effort.

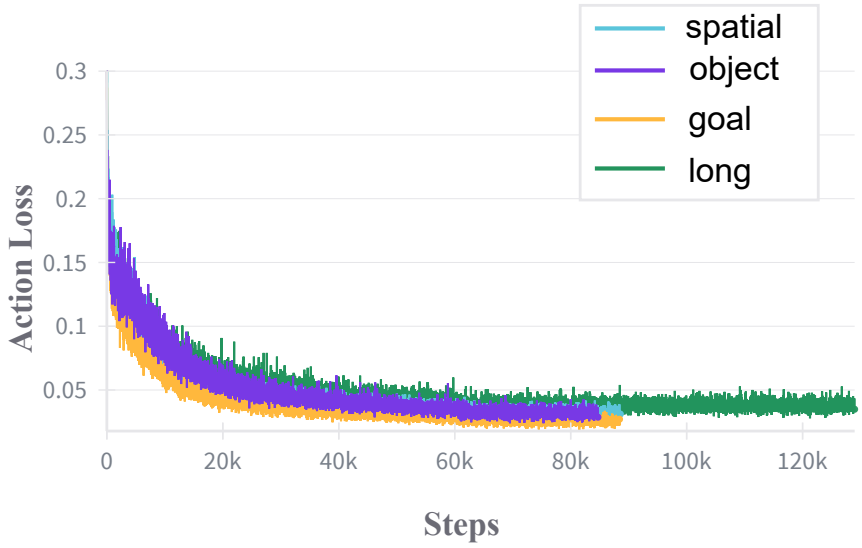

Figure 5: Training Loss Across LIBERO Datasets

Compare with OpenVLA-OFT Kim et al. (2025). The detailed parameters for libero benchmark are shown in Table 9. Hyperparameters for OpenVLA-OFT fine-tuning in Table 10. OpenVLA-OFT finetuned on OpenVLA model using 150 K training steps on 8 A100 GPUs, with a per-GPU batch size of 8. We define training effort as the total number of samples trained (training steps multiplied by the global batch size). Our model was trained with a global batch size of 40, while the OpenVLA-OFT used a larger batch size of 64. In comparison, our model requires less than 20% of the training effort for the object, spatial, and goal tasks, and 54% of the training effort for long-horizon tasks.

C Environments

C.1 Software Environment

Operating System: Ubuntu 22.04

Our model is implemented with:

- **Python:** 3.10.18
- **PyTorch:** 2.3.1
- **TorchVision:** 0.18.1
- **Transformers:** 4.51.0

In `SimplerEnv` benchmark. The key software dependencies are as follows:

- **TensorFlow:** 2.15.0
- **NumPy:** 1.24.4

C.2 Edge Computing Environment

The NVIDIA AGX Orin specifications are shown in Table 11.

Google Robot	Method	Pick Coke Can	Move Near	Open/Close Drawer	Avg.	Latency (ms) ↓	Speedup ↑
SimplerEnv (Visual Matching)	RT-1-X	56.7	31.7	59.7	49.4	–	–
	RT-2-X	78.7	77.9	25.0	60.5	–	–
	Octo-Base	17.0	4.2	22.7	14.6	–	–
	OpenVLA	18.0	56.3	63.0	34.3	240	1.0
	HPT	56.0	60.0	24.0	46.0	–	–
	TraceVLA	28.0	53.7	57.0	42.0	–	–
	RoboVLM (zero-shot)	72.7	66.3	26.8	56.3	–	–
	RoboVLM (fine-tuned)	77.3	61.7	43.5	63.4	–	–
	π_0	72.7	65.3	38.3	58.8	–	–
	π_0-FAST	75.3	67.5	42.9	61.9	470	0.5
	SpatialVLA (zero-shot)	81.0	69.6	59.3	70.0	400	0.6
	SpatialVLA (fine-tuned)	86.0	77.9	57.4	73.8	400	0.6
	CogACT	91.3	85.0	71.8	82.7	220	1.1
	Ours	89.0	78.8	56.9	74.9	78	3.1
SimplerEnv (Variant Aggregation)	RT-1	89.8	50.0	32.3	43.7	–	–
	RT-1-X	49.0	32.3	29.4	36.9	–	–
	RT-2-X	82.3	79.2	35.3	65.6	–	–
	Octo-Base	0.6	3.1	1.1	1.6	–	–
	OpenVLA	60.8	67.7	28.8	39.3	240	1.0
	TraceVLA	60.0	56.4	31.0	45.0	–	–
	RoboVLM (zero-shot)	68.3	56.0	8.5	46.3	–	–
	RoboVLM (fine-tuned)	75.6	60.0	10.6	51.3	–	–
	π_0	75.2	63.7	25.6	54.8	–	–
	π_0-FAST	77.6	68.2	31.3	59.0	470	0.5
	SpatialVLA (zero-shot)	89.5	71.7	36.2	65.8	400	0.6
	SpatialVLA (fine-tuned)	88.0	72.7	41.8	67.5	400	0.6
	CogACT	89.6	80.8	28.3	66.2	220	1.1
	Ours	84.3	82.5	13.8	60.2	78	3.1

Table 8: Comparison of our approach with existing VLA models on the Google robot across three tasks in two SimplerEnv settings. OpenVLA success rate is reported in CogACT Li et al. (2024a). The zero-shot and fine-tuning results denote performance of OXE datasetO'Neill et al. (2024) pre-trained models and Fractal datasetBrohan et al. (2022) fine-tuned models, respectively.

Hyperparameter	**Value**
# GPUs	2 × NVIDIA H100 (94GB VRAM)
Learning rate (LR)	1e-4
Total batch size	40 (20 per GPU)
# Chunk8 Train steps	10K (object); 50K (goal, spatial); 130K (long, with LR=1e-5 after 60K steps)
# Chunk16 Train steps	40K (goal); 20K (object, spatial); 80 K (long)
Input images	1 third-person camera image
Input image size	224 × 224 px
Use observation history	No (use single-step inputs)
LoRA rank	32
Action chunk size	8/16 steps
# Trainable parameters	148M total (111M LoRA adapter + 37M action head)

Table 9: Hyperparameters for LIBERO experiments.

Hyperparameter	**Value**
# GPUs	8 × NVIDIA A100 or H100 (80GB VRAM)
Learning rate (LR)	5e-4
Total batch size	64 (8 per GPU)
# Train steps	150K for LIBERO-Spatial (with LR=5e-5 after 100K steps); 150K for LIBERO-Object (with LR=5e-5 after 100K steps); 50K for LIBERO-Goal; 150K for LIBERO-Long (with LR=5e-5 after 100K steps)
Input images	1 third-person camera image
Input image size	224 × 224 px
Use observation history	No (use single-step inputs)
LoRA rank	32
Action chunk size	8 steps
# Trainable parameters	262M total (111M LoRA adapter + 151M action head)

Table 10: OpenVLA-OFT hyperparameters for LIBERO.Kim et al. (2025)

GPU	NVIDIA Ampere architecture; 2 GPCs, 8 TPCs, 16 SMs; 2048 CUDA cores (128 per SM); 64 Tensor Cores; 192KB L1 cache per SM; 4MB L2 cache
CPU	12-core Arm Cortex-A78AE v8.2 (64-bit), organized in 3 clusters; 64KB L1i/L1d per core; 3MB L2 (256KB/core); 6MB L3 (2MB/cluster); 4MB system cache
Memory	Unified 32GB LPDDR5 (256-bit), 204.8 GB/s bandwidth
Storage	4TB NVMe SSD and 32GB eMMC 5.1
Power	Up to 60W

Table 11: NVIDIA AGX Orin Specifications