

VLASH: Real-Time VLAs via Future-State-Aware Asynchronous Inference

Jiaming Tang^{1,*} Yufei Sun^{1,3,*} Yilong Zhao⁴ Shang Yang¹ Yujun Lin²
 Zhuoyang Zhang¹ James Hou^{1,6} Yao Lu² Zhijian Liu^{2,5} Song Han^{1,2}

¹MIT ²NVIDIA ³Tsinghua University ⁴UC Berkeley ⁵UCSD ⁶Caltech

<https://github.com/mit-han-lab/vlash>



Figure 1. **VLASH enables VLA to play ping-pong rallies with humans.** Snapshots showing $\pi_{0.5}$ [16] with VLASH successfully tracking and striking a fast-moving ping-pong ball during a rally. The robot initiates its reaction by the third frame, demonstrating low-latency perception-to-action response. The task requires both fast reaction and smooth continuous motion, which are enabled by our asynchronous inference with *future-state-awareness*. Under synchronous inference, the robot fails to achieve this dynamic interaction.

Abstract

Vision-Language-Action models (VLAs) are becoming increasingly capable across diverse robotic tasks. However, their real-world deployment remains slow and inefficient: demonstration videos are often sped up by 5-10 \times to appear smooth, with noticeable action stalls and delayed reactions to environmental changes. **Asynchronous inference** offers a promising solution to achieve continuous and low-latency control by enabling robots to execute actions and perform inference simultaneously. However, because the robot and environment continue to evolve during inference, a **temporal misalignment** arises between the prediction and execution intervals. This leads to significant action instability, while existing methods either degrade accuracy or introduce runtime overhead to mitigate it. We propose VLASH, a general asynchronous inference framework for VLAs that delivers smooth, accurate, and fast reaction control without additional overhead or architectural changes. VLASH estimates the future execution-time state by rolling the robot state forward with the previously generated action chunk, thereby bridging the gap between prediction and execution. Experiments show that VLASH achieves up to 2.03 \times speedup and reduces reaction latency by up to 17.4 \times compared to syn-

chronous inference while fully preserving the original accuracy. Moreover, it empowers VLAs to handle fast-reaction, high-precision tasks such as playing ping-pong and playing whack-a-mole, where traditional synchronous inference fails.

1. Introduction

Recent advances in Vision-Language-Action models (VLAs) such as $\pi_{0.5}$ [16], Gemini [1, 34] and Gr00t [26] have demonstrated remarkable capabilities in solving complex robotic tasks. In real-world deployment, these models are typically executed under a *synchronous inference* paradigm: the robot first performs model inference to generate an action chunk [41], then sequentially executes the actions before initiating the next inference cycle. This sequential pipeline introduces action stalls and delayed reactions to environmental changes, since the model remains idle during action execution and cannot update its perception in real time [4]. As a result, many VLA demonstration videos are sped up by several times to mask the discontinuous and slow motion.

To prevent this stop-and-go behavior, researchers have proposed *asynchronous inference* [4, 24, 29, 31]. In a nutshell, asynchronous inference allows the robot to execute the current action chunk while simultaneously performing

* indicates equal contributions.

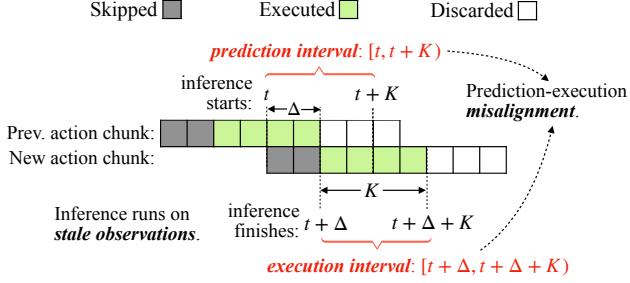


Figure 2. Prediction-execution misalignment in asynchronous inference. Due to inference delay Δ , the model predicts actions for the *prediction interval* $[t, t + K]$ but they execute during the *execution interval* $[t + \Delta, t + \Delta + K]$.

inference for the next one. Because the execution duration of an action chunk is typically longer than the model inference time, the robot can immediately switch to the next chunk once the inference completes, avoiding idle period between chunks [4, 24, 29, 31]. This design eliminates action stalls and allows the robot to perform smooth, continuous motion. Moreover, since inference is performed continuously, the robot can maintain real-time perception and thus react to environmental changes more promptly and accurately [4, 24]. In summary, asynchronous inference provides a promising way to achieve smooth, accurate, and fast reaction control for VLAs.

However, asynchronous inference faces a fundamental challenge that makes it unstable and inaccurate in practice. Since both the robot and the environment continue to evolve during inference, a *temporal misalignment* arises between the prediction interval starting when inference begins and the execution interval starting when inference finishes [4, 29]. As a result, the newly generated action misaligns with the robot’s execution-time state and environment, leading to severe instability and degraded control accuracy. For example, naive asynchronous inference reduces reaction latency but exhibits unstable and laggy control performance [4]. RTC [4] mitigates this by freezing the actions guaranteed to execute and inpainting the rest, but it introduces additional runtime overhead and complicates the deployment. In addition, current implementations [24, 29, 31] often require multi-threaded redesign of the inference framework to support asynchronous inference efficiently. Together, these create a significant barrier for the adoption of asynchronous inference for VLAs.

To address these challenges, we propose VLASH, a *general* asynchronous inference framework for VLAs that achieves *smooth, accurate, and fast reaction control without additional overhead or architectural changes*. In a nutshell, VLASH makes the model *future-state-aware* by accurately estimating the execution-time robot state using the previously issued action chunk, effectively *bridging the gap between prediction and execution*.

VLASH integrates seamlessly into existing fine-tuning pipelines and introduces no additional cost or latency. With a clean and lightweight implementation, VLASH provides a full-stack asynchronous inference framework from fine-tuning to inference at deployment, making asynchronous control practical and easy to adopt for real-time VLA systems.

We build and evaluate VLASH across various VLA models, including $\pi_{0.5}$ [16] and SmolVLA [31]. On simulation benchmarks [25], VLASH achieves up to 30.5% accuracy improvement compared to naive asynchronous inference and consistently outperforms all baselines. On real-world benchmarks [31], VLASH achieves up to 2.03 \times speedup and reduces reaction latency by up to 17.4 \times compared to synchronous inference while fully preserving the original accuracy. Beyond quantitative gains, VLASH demonstrates that large VLA models can handle fast-reaction, high-precision tasks such as playing ping-pong and playing whack-a-mole, which were previously infeasible under synchronous inference. We hope these results will inspire future research toward extending VLAs to more dynamic and physically interactive robotics.

2. Related Work

Vision-Language-Action Models (VLAs). Recent advances in Vision-Language-Action models have demonstrated remarkable capabilities in robotic manipulation by leveraging large-scale pretraining on diverse and internet-scale vision-language data. Models such as $\pi_{0.5}$ [16], RT-2 [43], and Gr0ot [26], etc. [3, 19] combine visual encoders with large language models to enable generalist robotic policies that can follow natural language instructions and generalize across tasks and embodiments. These models are typically deployed under synchronous inference, where the robot waits for model inference to complete before executing actions, resulting in action stall and slow reaction to environmental changes [4, 29]. Our work addresses this limitation by enabling efficient asynchronous inference for VLAs.

Asynchronous VLA Inference. Asynchronous inference offers a promising way to eliminate action stalls and improve reaction speed of VLAs, but existing approaches still face significant barriers to adoption in VLA community. SmolVLA [31] implements naive asynchronous inference by directly switching to new action chunks, but this causes severe prediction-execution misalignment and unstable control. Real-time Chunking (RTC) [4] mitigates this by freezing actions guaranteed to execute and inpainting the remaining actions, but this introduces additional runtime overhead for the inpainting process and complicates deployment. A concurrent work A2C2 [29] adds an additional correction heads to the model to mitigate the prediction-execution misalignment, but this also introduces runtime overhead and

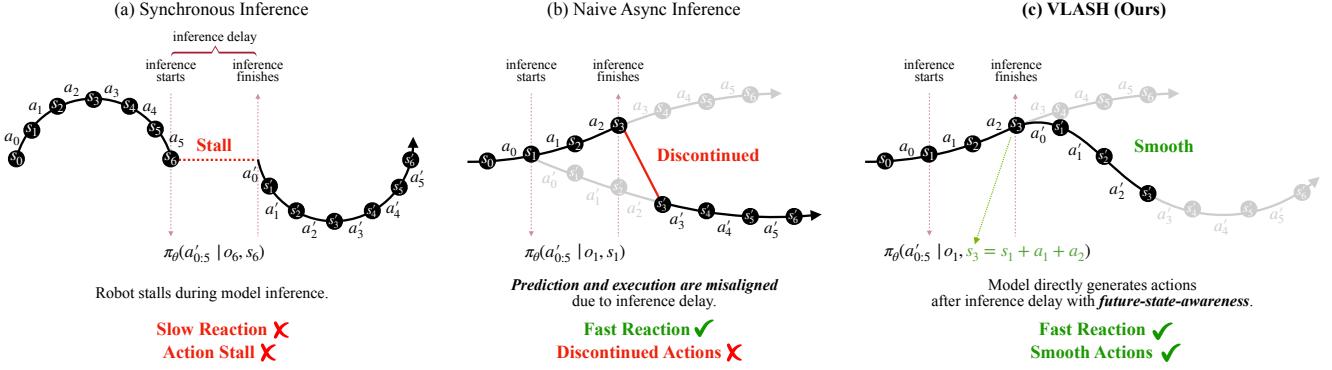


Figure 3. **Comparison between VLASH and existing methods.** (a) Synchronous inference: the robot stalls during inference, introducing slow reactions. (b) Naive async: the model predicts based on stale state s_1 while execution begins at future state s_3 , causing misalignment and discontinuity. (c) VLASH rolls forward the robot state ($s_3 = s_1 + a_1 + a_2$) and condition on the execution-time state, achieving fast reaction and smooth actions.

requires architecture changes to the model. In contrast, our method achieves asynchronous inference through future-state-awareness without additional overhead.

3. Background

Action chunking policy. We consider an *action chunking policy* $\pi_\theta(A_t | o_t, s_t)$ [16, 31, 42], where o_t is the environment observation (e.g., image, multi-view visual input), s_t is the robot state (e.g., joint positions, gripper state), and t is the controller timestep. At each timestep t , the policy generates a chunk of future actions

$$A_t = [a_t, a_{t+1}, \dots, a_{t+H-1}],$$

where H is the number of actions in the chunk. We refer to H as the *prediction horizon*.

Prediction and execution intervals. In practice, only the first $K \leq H$ actions from each chunk are executed before the next inference to ensure control accuracy. We denote K as the *execution horizon*. For a chunk A_t predicted at timestep t , we define the *prediction interval*

$$I_t^{\text{pred}} = [t, t + K]$$

as the time interval where the first K actions from the action chunk A_t are *planned* to be executed. During actual execution, however, the K actions from A_t will start being applied later due to inference latency [4, 31].

Let $\Delta > 0$ be the inference latency measured in control steps. Then the K actions from A_t are actually executed on the robot over the *execution interval*

$$I_t^{\text{exec}} = [t + \Delta, t + \Delta + K].$$

Asynchronous inference and interval misalignment. With asynchronous inference, the robot continues executing the previous action chunk while π_θ computes A_t in the background. As illustrated in Fig. 2, when $\Delta > 0$, the action chunk A_t is *planned* for the prediction interval $I_t^{\text{pred}} = [t, t + K]$ but actually *executed* over the shifted execution interval $I_t^{\text{exec}} = [t + \Delta, t + \Delta + K]$. Intuitively, the actions in A_t are not wrong for the original prediction interval $[t, t + K]$. However, under asynchronous inference, by the time they are executed, the environment and robot state have changed, so the same action sequence is applied to a different state and scene, leading to unstable and discontinuous behavior [4, 29].

4. VLASH

4.1. Future State Awareness

In asynchronous inference, the robot keeps moving while the VLA performs a forward pass, so the state at inference start generally differs from the state at which the new actions actually begin execution. Our key idea is to make the policy *future-state-aware*: instead of conditioning on the current robot state s_t , we condition on the robot state at the beginning of the next execution interval $s_{t+\Delta}$.

Although the future environment observation is unknown, the robot state at the beginning of the execution interval $s_{t+\Delta}$ is determined by the current robot state s_t and the actions executed during the inference delay $a_{t:t+\Delta-1}$. As shown in Fig. 3(c), when inference for the new chunk starts at state s_1 , the robot will still execute the remaining actions a_1, a_2 from the previous chunk before the new chunk is ready to take over. Since the actions a_1, a_2 are already known, we can *roll the state forward* under them to obtain the execution-time state. In the Fig. 3(c), this corresponds to computing $s_3 = s_1 + a_1 + a_2$, which gives the robot state

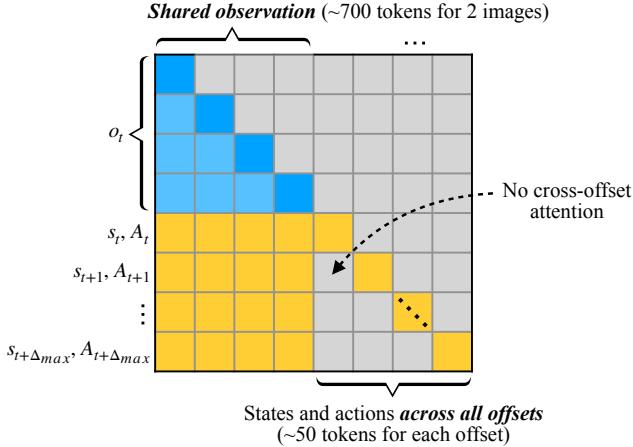


Figure 4. Attention pattern for efficient fine-tuning with shared observation. We pack one shared observation o_t and multiple offset branches $(s_{t+\delta}, A_{t+\delta})$ into a single sequence. Blue and yellow cells indicate allowed attention, while gray cells indicate masked attention. Positional encodings of each offset branch are reassigned to start at the same index, equal to the length of observation tokens.

at the start of the execution interval.

During the forward pass, VLASH feeds both the current environment observation o_1 and this rolled-forward future state s_3 into the VLA. In this way, the model generates actions for the state at the execution-time rather than for the stale state at inference start, *bridging the gap between prediction and execution in terms of robot state*. While the future environment is still unknown, this mechanism mirrors how humans act under reaction delays: we react to the world with slightly outdated visual input, but use our internal body state to anticipate what we will do when the action actually takes effect. Thus, humans inherently have the ability to compensate for such reaction delay, and we expect VLAs to possess the same capability.

4.2. Fine-tuning with Offsets to States and Actions

The future-state-awareness assumes that the VLA is able to leverage the rolled-forward robot state. However, we find that existing VLAs often fail to exploit this future state properly. Even more, current VLAs appear to largely *rely on visual input and under-utilize the robot state*. In our experiments with $\pi_{0.5}$ (Table 1), fine-tuning *without* state input (visual only) consistently outperforms fine-tuning *with* state input on LIBERO [23]. Therefore, simply feeding a future robot state at test time is insufficient to achieve accurate and stable asynchronous control.

Since large VLAs are almost always fine-tuned on downstream data before deployment, we design a training augmentation that can be seamlessly integrated into the standard fine-tuning stage with *no additional overhead*. We keep the architecture and fine-tuning pipeline unchanged, and only

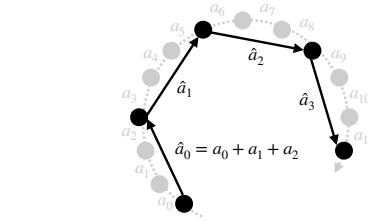


Figure 5. Action quantization for efficient execution. We group consecutive *fine-grained* micro-actions into *coarser* macro-actions to accelerate robot motion. The original trajectory with fine-grained actions a_0, a_1, a_2, \dots (gray) is quantized into a shorter trajectory with macro-actions $\hat{a}_0, \hat{a}_1, \hat{a}_2, \hat{a}_3$ (black), where each macro-action summarizes q consecutive fine-grained actions (e.g., $\hat{a}_0 = a_0 + a_1 + a_2$ for quantization factor $q = 3$).

modify how training samples are constructed.

Concretely, given a trajectory $\{(o_t, s_t, a_t)\}$, standard fine-tuning trains the model to predict the action chunk $a_{t:t+H-1}$ from (o_t, s_t) . We instead apply a simple temporal-offset augmentation with two key steps:

- (i) **Offset state and action together.** We sample a random offset δ from a predefined range (e.g., $\delta \in \{0, \dots, \Delta_{\max}\}$) and construct training targets from the future state $s_{t+\delta}$ and future action chunk $a_{(t+\delta):(t+\delta+H-1)}$ on the same trajectory.
- (ii) **Fix the environment observation.** For each timestep t , we always use the same visual input o_t when varying δ . Therefore, the model is trained to predict $a_{(t+\delta):(t+\delta+H-1)}$ from the pair $(o_t, s_{t+\delta})$.

Under this scheme, the same image o_t can correspond to different ground-truth actions depending on the offset robot state $s_{t+\delta}$. To fit the data, the VLA is forced to attend to the state input rather than overfitting purely to visual features. In particular, it learns to interpret $s_{t+\delta}$ as a meaningful future state for action selection.

We randomly sample δ during training because, in practice, the same VLA may be deployed on hardware with different compute budgets, leading to different inference delays Δ , and sometimes even in synchronous settings where there is no gap between prediction and execution. By training over a range of offsets, our augmentation makes the model *compatible with different inference delays* while preserving performance in the synchronous case. At deployment with asynchronous inference, we can then feed the rolled-forward execution-time state together with the current observation, and the fine-tuned VLA naturally leverages this future state to produce actions that are aligned and stable over the execution interval.

4.3. Efficient Fine-tuning with Shared Observation

The temporal-offset augmentation creates multiple state-action pairs for the same observation o_t . A naive implementation would treat each offset δ as a separate training ex-

ample, i.e., run the VLA independently on $(o_t, s_{t+\delta}, A_{t+\delta})$ for each sampled δ . This implementation is completely plug-and-play and can be seamlessly integrated into existing VLA fine-tuning pipeline. However, it repeatedly encodes the same observation o_t for every offset, leaving substantial room for further efficiency gains.

Instead, we exploit the fact that all offsets share the same observation o_t and design an efficient attention pattern that reuses the observation tokens across offsets in a single pass (Fig. 4). Concretely, we pack one observation and multiple offset branches into a single sequence:

$$[o_t, (s_t, A_t), (s_{t+1}, A_{t+1}), \dots, (s_{t+\Delta_{\max}}, A_{t+\Delta_{\max}})],$$

where each $(s_{t+\delta}, A_{t+\delta})$ corresponds to one temporal offset. We then apply a block-sparse self-attention mask with the following structure:

- All **observation tokens** (e.g., image tokens from two views and language prompt, about ~ 700 tokens for $\pi_{0.5}$) can attend to each other, as in standard VLA fine-tuning.
- For each **offset branch**, the state-action tokens $(s_{t+\delta}, A_{t+\delta})$ can attend to all observation tokens and to tokens within the same offset, but *cannot* attend to tokens from other offsets.

This attention map, illustrated in Fig. 4, makes different offsets condition on a shared observation while remaining independent of each other. For each offset branch, the **positional encodings** of $(s_{t+\delta}, A_{t+\delta})$ are assigned to start at the same index, equal to the length of observation tokens. From the model’s perspective, this is equivalent to training on multiple $(o_t, s_{t+\delta}, A_{t+\delta})$ examples that share the same o_t , but we only encode o_t once.

For $\pi_{0.5}$, an observation with two images and language prompt corresponds to ~ 700 tokens, while one state and action chunk are about ~ 50 tokens [16]. Therefore, packing $N_\delta = 5$ offsets into a single sequence therefore increases the token length by only $\sim 20\%$, while the number of effective training trajectories becomes $5\times$ larger. In practice, under the same *effective batch size* as standard fine-tuning, this method can significantly improve training efficiency by reusing each observation across multiple offset targets in a single pass.

4.4. Action Quantization

With asynchronous inference and future-state-awareness, the model inference time is effectively hidden behind execution. Once this inference latency is removed, the overall speed of the system is primarily limited by how fast the robot can physically execute the action sequence. To push the execution speed further, we need to accelerate the motion itself.

Our approach is to *quantize actions*, in analogy to weight quantization for LLMs [11, 22, 37]. State-of-the-art VLAs are typically trained on fine-grained teleoperation data (e.g.,

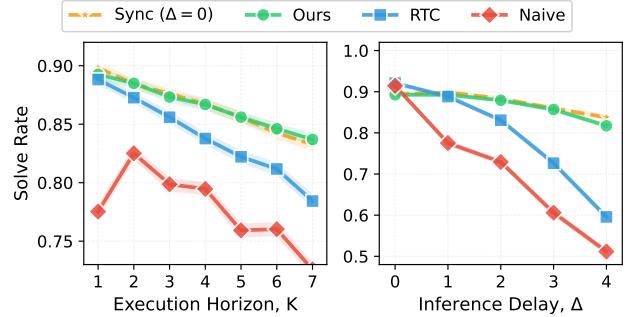


Figure 6. **Performance on Kinetix benchmark.** We evaluate the success rate under different execution horizons K and inference delays Δ . **Left:** Fixed inference delay $\Delta = 1$ with varying execution horizon K . **Right:** Execution horizon adapts to inference delay, i.e., $K = \max(\Delta, 1)$, with varying Δ . For the **Sync** baseline, inference delay is always $\Delta = 0$, but the execution horizon K follows the same settings as other baselines for fair comparison.

~ 50 Hz control with small deltas at each step) [3, 16], which leads to action sequences with high granularity. However, many short micro-movements are more precise than what is actually required to solve the tasks. In LLMs, 16-bit weights provide high numerical precision, but quantizing them to 8-bit or 4-bit can substantially accelerate inference with only a mild drop in accuracy [11, 22, 37]. We apply the same philosophy to robot control.

Given a fine-grained action sequence $\{a_0, a_1, \dots, a_T\}$, we group consecutive actions into coarser *macro-actions*. For a chosen quantization factor q , we construct a new sequence $\{\hat{a}_0, \hat{a}_1, \dots\}$ where each macro-action summarizes a block of q fine-grained actions. For delta actions, this can be implemented as

$$\hat{a}_i = a_{iq} + a_{iq+1} + \dots + a_{(i+1)q-1}$$

so that \hat{a}_i takes the robot approximately from the start state of a_{iq} to the end state of $a_{(i+1)q-1}$ in a single, longer step. Fig. 5 illustrates this process: the original fine-grained trajectory (gray) is replaced by a shorter, quantized trajectory (black) with macro-actions $\hat{a}_0, \hat{a}_1, \hat{a}_2, \hat{a}_3$, where $\hat{a}_0 = a_0 + a_1 + a_2$.

Executing macro-actions instead of all micro-actions increases the distance moved per control step, effectively speeding up the robot’s motion. The temporal granularity of control becomes coarser, but in many tasks the robot does not need to visit every intermediate waypoint explicitly; moving directly between sparser waypoints is sufficient to achieve the goal. As a result, action quantization offers a tunable speed-accuracy trade-off: small quantization factors behave like the original fine-grained policy, while larger factors yield progressively faster but less fine-grained motion. In practice, we select task-dependent quantization factors that maintain success rates close to the unquantized policy while substantially reducing the number of executed steps.

Table 1. **Performance on LIBERO benchmarks with different inference delays.** We evaluate $\pi_{0.5}$ [16] across four LIBERO sub-benchmarks (Spatial, Object, Goal, LIBERO-10) under various inference delays (0 to 4 steps). **SR**: average success rate; **Steps**: average execution steps to task completion; **Time**: completion time on a laptop RTX 4090 GPU (inference latency: 103ms for 2 images). **Sync (w/o state)**: fine-tuned and evaluated with synchronous inference without robot state input.

Method	Delay	Success Rate (%)				Average			Improvement	
		Spatial	Object	Goal	LIBERO-10	SR	Steps	Time (s)	Δ SR	Speedup
Sync	0	97.3	99.6	96.7	93.5	96.8	156.0	8.4	-	-
Sync (w/o state)	0	98.5	99.6	97.3	95.4	97.7	157.2	8.4	+0.9	-
VLASH (Async)	1	98.8	99.2	96.7	94.4	97.2	153.9	7.2	+0.4	1.17×
	2	97.5	99.2	97.3	94.6	97.1	157.6	6.4	+0.3	1.31×
	3	94.4	98.8	93.3	91.9	94.6	167.3	5.7	-2.2	1.47×
	4	92.5	96.9	93.3	89.6	93.1	176.7	5.8	-3.7	1.45×

5. Experiments

We design experiments to investigate the following questions:

1. **Performance.** How does our method compare to synchronous control, naive asynchronous and baselines in terms of accuracy and latency? (Sec. 5.1.1, Sec. 5.2)
2. **Generalization.** How well does our method generalize across different inference delays? Does it hurt the original model performance? How well does our method generalize across different VLAs? (Sec. 5.1.2)
3. **Speed-accuracy trade-off.** What is the speed-accuracy trade-off of action quantization at deployment? (Sec. 5.2)
4. **Fine-tuning efficiency.** How does our method compare to the standard fine-tuning in terms of training cost and data efficiency? How much the shared observation fine-tuning can reduce the training cost? (Sec. 5.3)

5.1. Simulated Evaluation

We evaluate VLASH on simulated robotic manipulation benchmarks including Kinetix [25] and LIBERO [23].

5.1.1. Kinetix

Experimental Setup. Kinetix [25] is a highly dynamic simulated robotic manipulation benchmark that demands asynchronous execution to handle rapidly changing environments. The tasks are designed to test dynamic reaction capabilities, including throwing, catching, and balancing.

Following the setup in RTC [4], we train action chunking flow policies with a prediction horizon of $H = 8$ and a 4-layer MLP-Mixer [35] architecture for 32 epochs. We report average success rates across 12 tasks, each evaluated with 1,024 rollouts per data point, under simulated delays ranging from 0 to 4 steps. We compare against the following baselines:

- **Sync.** This baseline serves as an optimal baseline for all tasks. The **inference delay is explicitly set to 0** at all times.

- **Naive async.** This baseline is the naive asynchronous inference baseline, which simply switches chunks as soon as the new one is ready [31].
- **RTC.** This baseline is the Real-time Chunking [4], which freezes the actions guaranteed to execute and inpaints the rest. This introduces additional overhead at runtime.

Results. As shown in Fig. 6, VLASH tracks the synchronous upper bound closely across execution horizons, while other baselines drop more noticeably as the execution horizon increases. When the inference delay increases, VLASH remains robust and consistently achieves high success rates, while RTC degrades rapidly and the Naive Async baseline collapses under larger delays. Notably, at inference delay of 4 steps, VLASH achieves 81.7% success rate compared to only 51.2% for Naive Async, which is a substantial 30.5% accuracy improvement. Overall, VLASH effectively mitigates prediction-execution misalignment, delivering high success rates under asynchronous operation.

5.1.2. LIBERO

Experimental Setup. We evaluate on LIBERO benchmark [23], one of the popular benchmarks for evaluating VLA, which includes 4 different sub-benchmarks (Spatial, Object, Goal, and LIBERO-10) that contain 10 tasks each. We evaluate on 2 state-of-the-art VLAs: $\pi_{0.5}$ [16] and SmoVLA [31]. We report the performance by fine-tuning all models on the training dataset for 30K iterations with a batch size of 32. Following the setup in $\pi_{0.5}$ [16], we set the execution horizon to $K = 5$ [10]. Since LIBERO tasks involve slowly changing environments with mild state transitions, different asynchronous methods behave similarly. Therefore, we focus our comparisons on synchronous inference to evaluate the effectiveness of VLASH under various inference delays. For time measurement, we use a laptop RTX 4090 GPU where the inference latency with 2 input images is 103ms. For synchronous inference, the time per action chunk is the sum of execution duration (166ms for

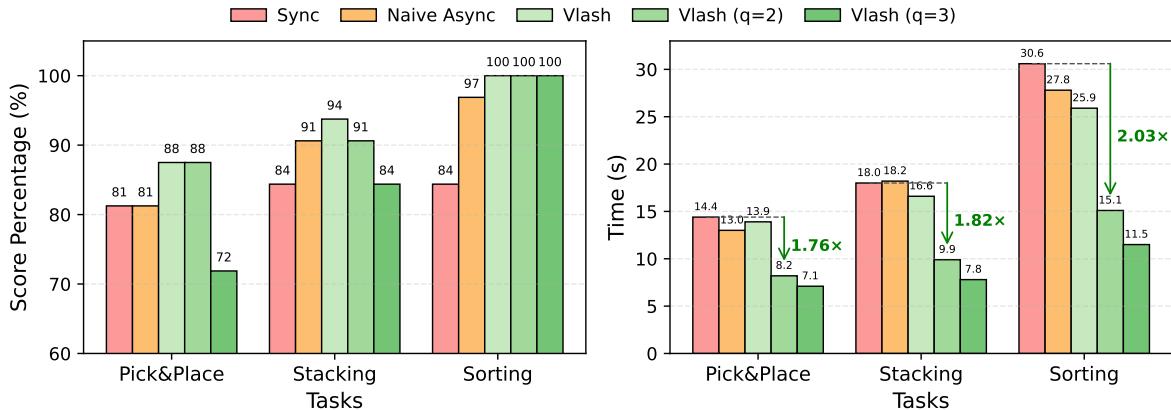


Figure 7. **Real-world evaluation results on manipulation tasks.** We evaluate $\pi_{0.5}$ [16] on three tasks with different inference methods. **Left:** Score percentages (based on 2-point scoring: 1 for success of picking up the object, 1 for task completion) of VLASH and baselines across three tasks. **Right:** Task completion times with green arrows indicating speedup of VLASH ($q=2$) relative to synchronous baseline. VLASH (q) applies action quantization with quantization ratio q .

$K = 5$ steps at 30Hz) and inference time. For asynchronous inference, larger delays are needed to overlap with the inference latency, so the time per action chunk is: execution duration + max(0, inference time - $\frac{\text{execution duration}}{K} \times \text{delay}$).

Results. As shown in Table 1, VLASH demonstrates strong performance across all LIBERO benchmarks under various inference delays. With small inference delays, VLASH maintains comparable accuracy to synchronous inference while achieving speedups of 1.17 \times and 1.31 \times , respectively. As the inference delay increases, the time advantages become more pronounced, achieving up to 1.47 \times speedup at delay 3. Although accuracy decreases slightly at higher delays, VLASH still achieves strong performance across all tasks, demonstrating an effective accuracy-latency trade-off. We also evaluate on SmoVLA [31], with detailed results provided in supplementary materials.

5.2. Real-World Evaluation

To evaluate VLASH in real-world settings, we deploy $\pi_{0.5}$ [16] on two robotic platforms: the Galaxeal R1 Lite [13] and the LeRobot SO-101 [15]. The R1 Lite is a dual-arm robot equipped with two 7-DOF arms from Galaxeal [12]. The SO-101 is a 6-DOF collaborative robotic arm from LeRobot [5]. For $\pi_{0.5}$, we apply a projection layer to map the robot state into an embedding, bypassing the tokenizer instead of incorporating it into the language prompt in the original implementation. We design our real-world experiments to evaluate three key aspects: (1) Accuracy: the success rate of completing manipulation tasks; (2) Efficiency: the task completion time and motion smoothness; and (3) Reaction speed: the latency to react to dynamic changes in the environment.

5.2.1. Accuracy and Efficiency

Experimental Setup. Following the setup in SmoVLA [31], we evaluate $\pi_{0.5}$ ($H = 50$) on three manipulation tasks that test different aspects of robotic control. We set the execution horizon to $K = 24$ steps at 30Hz. All experiments are conducted on a laptop with NVIDIA RTX 4090 GPU, with an inference delay of 4 steps. On our robotic platforms, we evaluate three tasks:

- **Pick and Place:** pick up a cube from varying starting positions and place it into a fixed box;
- **Stacking:** pick up a blue cube and stack it on top of an orange cube, where both cubes' initial positions vary across episodes;
- **Sorting:** sort cubes by color, placing the orange cube in the left box and the blue cube in the right box, with cube positions varying across episodes.

For each task, we conduct 16 rollouts per method and report both the score percentage and the task completion time. The score percentage is calculated based on a 2-point scoring system per rollout: 1 point for successfully picking up the object, and 1 point for completing the task. We compare synchronous inference, naive asynchronous inference, and VLASH across these tasks.

Results. As shown in Fig. 7, VLASH delivers better or comparable score percentage to synchronous inference while significantly reducing task completion time across all tasks. Specifically, VLASH maintains an 94% average score percentage, outperforming synchronous baseline (83%) and naive asynchronous inference (89.7%), while completing tasks in 18.8 seconds on average compared to 21 seconds for synchronous inference, which is a 1.12 \times speedup.

Furthermore, by applying action quantization, we can

Table 2. **Reaction speed comparison across devices.** Latency of $\pi_{0.5}$ [16] with 1 image input, $K = 25$ at 50Hz. Execution duration is 500ms. Max reaction latency = execution duration + inference latency for Sync, inference latency only for Async.

Inference Delay	RTX 5090	RTX 4090	RTX 5070
(in ms)	30.4	36.1	64.1
(in action steps)	1.52	1.81	3.21
Reaction (ms)			
Sync	530.4	536.1	564.1
Async	30.4	36.1	64.1
Speedup	17.4×	14.9×	8.8×

achieve greater speedups with minimal accuracy loss. VLASH with $q=2$ achieves up to $2.03\times$ speedup, while maintaining the original accuracy. With a more aggressive quantization ratio of $q=3$, VLASH achieves the faster execution at up to $2.67\times$ speedup, with only a modest 4.7% drop in average score percentage, which demonstrates a favorable speed-accuracy trade-off.

5.2.2. Reaction Speed

Experimental Setup. To evaluate the reaction speed improvement of asynchronous inference, we compare the maximum reaction latency between synchronous and asynchronous inference across different hardware configurations. Following the setup in $\pi_{0.5}$ [16], we set the execution horizon to $K = 25$ for synchronous inference and a control frequency of 50Hz [4, 16], resulting in an execution duration of approximately 0.5 seconds per action chunk. We measure the model inference latency of $\pi_{0.5}$ on three different GPUs: RTX 5090, RTX 4090, and RTX 5070, using `torch.compile` to enable CUDAGraph optimization and kernel fusion for minimal latency [2].

Results. As shown in Table 2, asynchronous inference significantly reduces the maximum reaction latency compared to synchronous inference, achieving up to $17.4\times$ speedup. To showcase the fast reaction and smooth control capabilities of VLASH, we train $\pi_{0.5}$ to perform highly dynamic interactive tasks: playing ping-pong with a human and playing whack-a-mole. These tasks demand both rapid reaction to dynamic changes and smooth continuous motion to maintain control accuracy. To the best of our knowledge, we are the first to **demonstrate a VLA successfully playing ping-pong rallies with a human**. Under synchronous inference, the robot’s reaction is too slow to track the fast-moving ball, while VLASH enables real-time response and stable rallies. We encourage readers to **view the demo videos in the supplementary materials** to see the dynamic performance of VLASH in action.

Table 3. **Fine-tuning efficiency.** Original (without offset augmentation) vs VLASH (with offset augmentation and shared observation) on LIBERO with $\pi_{0.5}$ [16]. Training on 4×H100 GPUs using DDP, with effective batch size 16 per GPU (total 64). We report average LIBERO scores at different training steps. Both evaluated under synchronous inference.

Method	Time/Step (ms)	Fine-tuning Steps		
		10K	20K	30K
Original	420.99	94.1	97.1	96.8
VLASH	129.29	87.1	94.4	96.6
Speedup	3.26×	-	-	-

5.3. Fine-tuning Efficiency

Experimental Setup. We evaluate the training efficiency gains from our efficient fine-tuning with shared observation approach. A key consideration is that training with multiple temporal offsets using shared observation effectively increases the *effective batch size* by a factor equal to the number of offsets. Therefore, we compare our method against standard fine-tuning under the same *effective batch size* to ensure a fair comparison. Specifically, we conduct experiments on the LIBERO benchmark using $\pi_{0.5}$ [16] trained on 4×H100 GPUs with DDP [21]. For our method, we use $\Delta_{\max} = 3$ with a physical batch size of 4 per GPU, resulting in an effective batch size of 16 per GPU and 64 in global. The standard baseline uses a physical batch size of 16 per GPU to match this effective batch size. Both methods are trained for 10K, 20K, and 30K iterations, and we report the average success rate across all LIBERO tasks. We also measure the training time per forward-backward pass to quantify the speedup.

Results. As shown in Table 3, VLASH converges more slowly in the early stages but ultimately achieves comparable accuracy to standard fine-tuning. Although more training steps are needed for convergence, each step is significantly faster, achieving a $3.26\times$ speedup per step. This efficiency gain comes from encoding the shared observation only once and reusing it across all temporal offsets. Furthermore, since both methods are evaluated under synchronous inference, these results also demonstrate that VLASH does not hurt the original synchronous performance of the model.

6. Conclusion

We present VLASH, a general and efficient framework for enabling asynchronous inference in Vision-Language-Action models. By making the policy future-state-aware through simple state rollforward, VLASH effectively bridges the prediction-execution gap that has hindered asynchronous control. Experiments on both simulated and real-world

benchmarks demonstrate that VLASH achieves smooth, accurate, and fast-reaction control, consistently matching or surpassing the accuracy of synchronous inference while providing substantial speedups. Moreover, we demonstrate that VLAs can perform highly dynamic tasks such as playing ping-pong rallies with humans. We hope these results will inspire future research toward extending VLAs to more dynamic and physically interactive domains.

Acknowledgements

We thank MIT-IBM Watson AI Lab, Amazon and National Science Foundation for supporting this research. We thank NVIDIA for donating the DGX server.

References

- [1] Abbas Abdolmaleki, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Ashwin Balakrishna, Nathan Batchelor, Alex Bewley, Jeff Birmingham, Michael Bloesch, et al. Gemini robotics 1.5: Pushing the frontier of generalist robots with advanced embodied reasoning, thinking, and motion transfer. *arXiv preprint arXiv:2510.03342*, 2025. [1](#)
- [2] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshitij Kalmabkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, C. K. Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathews, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, page 929–947, New York, NY, USA, 2024. Association for Computing Machinery. [8](#)
- [3] 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. [2](#), [5](#)
- [4] Kevin Black, Manuel Y Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. *arXiv preprint arXiv:2506.07339*, 2025. [1](#), [2](#), [3](#), [6](#), [8](#)
- [5] Remi Cadene, Simon Alibert, Alexander Soare, Quentin Gallouedec, Adil Zouitine, Steven Palma, Pepijn Kooijmans, Michel Aractingi, Mustafa Shukor, Dana Aubakirova, Martino Russi, Francesco Capuano, Caroline Pascal, Jade Choghari, Jess Moss, and Thomas Wolf. Lerobot: State-of-the-art machine learning for real-world robotics in pytorch. <https://github.com/huggingface/lerobot>, 2024. [7](#)
- [6] Embodiment Collaboration, Abby O'Neill, Abdul Rehman, Abhinav Gupta, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, Albert Tung, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anchit Gupta, Andrew Wang, Andrey Kolobov, Anikait Singh, Animesh Garg, Aniruddha Kembhavi, Annie Xie, Anthony Brohan, Antonin Raffin, Archit Sharma, Arefeh Yavary, Arhan Jain, Ashwin Balakrishna, Ayzaan Wahid, Ben Burgess-Limerick, Beomjoon Kim, Bernhard Schölkopf, Blake Wulfe, Brian Ichter, Cewu Lu, Charles Xu, Charlotte Le, Chelsea Finn, Chen Wang, Chenfeng Xu, Cheng Chi, Chenguang Huang, Christine Chan, Christopher Agia, Chuer Pan, Chuyuan Fu, Coline Devin, Danfei Xu, Daniel Morton, Danny Driess, Daphne Chen, Deepak Pathak, Dhruv Shah, Dieter Büchler, Dinesh Jayaraman, Dmitry Kalashnikov, Dorsa Sadigh, Edward Johns, Ethan Foster, Fangchen Liu, Federico Ceola, Fei Xia, Feiyu Zhao, Felipe Vieira Frujeri, Freek Stulp, Gaoyue Zhou, Gaurav S. Sukhatme, Gautam Salhotra, Ge Yan, Gilbert Feng, Giulio Schiavi, Glen Berseth, Gregory Kahn, Guangwen Yang, Guanzhi Wang, Hao Su, Hao-Shu Fang, Haochen Shi, Henghui Bao, Heni Ben Amor, Henrik I Christensen, Hiroki Furuta, Homanga Bharadhwaj, Homer Walke, Hongjie Fang, Huy Ha, Igor Mordatch, Ilijia Radosavovic, Isabel Leal, Jacky Liang, Jad Abou-Chakra, Jaehyung Kim, Jaimyn Drake, Jan Peters, Jan Schneider, Jasmine Hsu, Jay Vakil, Jeannette Bohg, Jeffrey Bingham, Jeffrey Wu, Jensen Gao, Jiaheng Hu, Jiajun Wu, Jialin Wu, Jiankai Sun, Jianlan Luo, Jiayuan Gu, Jie Tan, Jihoon Oh, Jimmy Wu, Jingpei Lu, Jingyun Yang, Jitendra Malik, João Silvério, Joey Hejna, Jonathan Booher, Jonathan Tompson, Jonathan Yang, Jordi Salvador, Joseph J. Lim, Junhyek Han, Kaiyuan Wang, Kanishka Rao, Karl Pertsch, Karol Hausman, Keegan Go, Keerthana Gopalakrishnan, Ken Goldberg, Kendra Byrne, Kenneth Oslund, Kento Kawaharazuka, Kevin Black, Kevin Lin, Kevin Zhang, Kiana Ehsani, Kiran Lekkala, Kirsty Ellis, Krishan Rana, Krishnan Srinivasan, Kuan Fang, Kunal Pratap Singh, Kuo-Hao Zeng, Kyle Hatch, Kyle Hsu, Laurent Itti, Lawrence Yunliang Chen, Lerryl Pinto, Li Fei-Fei, Liam Tan, Linxi "Jim" Fan, Lionel Ott, Lisa Lee, Luca Weihs, Magnum Chen, Marion Lepert, Marius Memmel, Masayoshi Tomizuka, Masha Itkina, Matteo Guaman Castro, Max Spero, Maximilian Du, Michael Ahn, Michael C. Yip, Mingtong Zhang, Mingyu Ding, Minho Heo, Mohan Kumar Srirama, Mohit Sharma, Moo Jin Kim, Muhammad Zubair Irshad, Naoaki Kanazawa, Nicklas Hansen, Nicolas Heess, Nikhil J Joshi, Niko Suenderhauf, Ning Liu, Norman Di Palo, Nur Muhammad Mahi Shafullah, Oier Mees, Oliver Kroemer, Osbert Bastani, Pannag R Sanketi, Patrick "Tree" Miller, Patrick Yin, Paul Wohlhart, Peng Xu, Peter David Fagan, Peter Mitrano, Pierre Sermanet, Pieter Abbeel, Priya Sundaresan, Qiuyu Chen, Quan Vuong, Rafael Rafailov, Ran Tian, Ria Doshi, Roberto Martín-Martín, Rohan Baijal, Rosario Scalise, Rose Hendrix, Roy Lin, Runjia Qian, Ruohan Zhang, Russell Mendonca, Rutav Shah, Ryan Hoque, Ryan Julian, Samuel Bustamante, Sean Kir

- mani, Sergey Levine, Shan Lin, Sherry Moore, Shikhar Bahl, Shivin Dass, Shubham Sonawani, Shubham Tulsiani, Shuran Song, Sichun Xu, Siddhant Haldar, Siddharth Karamcheti, Simeon Adebola, Simon Guist, Soroush Nasiriany, Stefan Schaal, Stefan Welker, Stephen Tian, Subramanian Ramamoorthy, Sudeep Dasari, Suneel Belkhale, Sungjae Park, Suraj Nair, Suvir Mirchandani, Takayuki Osa, Tanmay Gupta, Tatsuya Harada, Tatsuya Matsushima, Ted Xiao, Thomas Kollar, Tianhe Yu, Tianli Ding, Todor Davchev, Tony Z. Zhao, Travis Armstrong, Trevor Darrell, Trinity Chung, Vidhi Jain, Vikash Kumar, Vincent Vanhoucke, Victor Guizilini, Wei Zhan, Wenxuan Zhou, Wolfram Burgard, Xi Chen, Xiangyu Chen, Xiaolong Wang, Xinghao Zhu, Xinyang Geng, Xiyuan Liu, Xu Liangwei, Xuanlin Li, Yangsong Pang, Yao Lu, Yecheng Jason Ma, Yejin Kim, Yevgen Chebotar, Yifan Zhou, Yifeng Zhu, Yilin Wu, Ying Xu, Yixuan Wang, Yonatan Bisk, Yongqiang Dou, Yoonyoung Cho, Youngwoon Lee, Yuchen Cui, Yue Cao, Yueh-Hua Wu, Yujin Tang, Yuke Zhu, Yunchu Zhang, Yunfan Jiang, Yunshuang Li, Yunzhu Li, Yusuke Iwasawa, Yutaka Matsuo, Zehan Ma, Zhuo Xu, Zichen Jeff Cui, Zichen Zhang, Zipeng Fu, and Zipeng Lin. Open x-embodiment: Robotic learning datasets and rt-x models, 2025.
- [7] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning, 2023.
- [8] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022.
- [9] Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. Flex attention: A programming model for generating optimized attention kernels, 2024.
- [10] Danny Driess, Jost Tobias Springenberg, Brian Ichter, Lili Yu, Adrian Li-Bell, Karl Pertsch, Allen Z. Ren, Homer Walke, Quan Vuong, Lucy Xiaoyang Shi, and Sergey Levine. Knowledge insulating vision-language-action models: Train fast, run fast, generalize better, 2025. 6
- [11] Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. GPTQ: Accurate post-training compression for generative pretrained transformers. *arXiv preprint arXiv:2210.17323*, 2022. 5
- [12] Ltd. Galaxea AI Technology Co. Galaxea ai. <https://galaxea-ai.com/cn>, 2025. 7
- [13] Ltd. Galaxea AI Technology Co. R1 lite. <https://galaxea-ai.com/cn/products/R1-Lite>, 2025. 7
- [14] Weifan Guan, Qinghao Hu, Aosheng Li, and Jian Cheng. Efficient vision-language-action models for embodied manipulation: A systematic survey. *arXiv preprint arXiv:2510.17111*, 2025.
- [15] Inc. Hugging Face. So-101. <https://huggingface.co/docs/lerobot/en/so101>, 2025. 7
- [16] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025. 1, 2, 3, 5, 6, 7, 8
- [17] Titong Jiang, Xuefeng Jiang, Yuan Ma, Xin Wen, Bailin Li, Kun Zhan, Peng Jia, Yahui Liu, Sheng Sun, and Xianpeng Lang. The better you learn, the smarter you prune: Towards efficient vision-language-action models via differentiable token pruning, 2025.
- [18] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, Peter David Fagan, Joey Hejna, Masha Itkina, Marion Lepert, Yecheng Jason Ma, Patrick Tree Miller, Jimmy Wu, Suneel Belkhale, Shivin Dass, Huy Ha, Arhan Jain, Abraham Lee, Youngwoon Lee, Marius Memmel, Sungjae Park, Ilija Radosavovic, Kaiyuan Wang, Albert Zhan, Kevin Black, Cheng Chi, Kyle Beltran Hatch, Shan Lin, Jingpei Lu, Jean Mercat, Abdul Rehman, Pannag R Sanketi, Archit Sharma, Cody Simpson, Quan Vuong, Homer Rich Walke, Blake Wulfe, Ted Xiao, Jonathan Heewon Yang, Arefeh Yavary, Tony Z. Zhao, Christopher Agia, Rohan Baijal, Mateo Guaman Castro, Daphne Chen, Qiuyu Chen, Trinity Chung, Jaimyn Drake, Ethan Paul Foster, Jensen Gao, Victor Guizilini, David Antonio Herrera, Minho Heo, Kyle Hsu, Jiaheng Hu, Muhammad Zubair Irshad, Donovan Jackson, Charlotte Le, Yunshuang Li, Kevin Lin, Roy Lin, Zehan Ma, Abhiram Maddukuri, Suvir Mirchandani, Daniel Morton, Tony Nguyen, Abigail O'Neill, Rosario Scalise, Derick Seale, Victor Son, Stephen Tian, Emi Tran, Andrew E. Wang, Yilin Wu, Annie Xie, Jingyun Yang, Patrick Yin, Yunchu Zhang, Osbert Bastani, Glen Berseth, Jeannette Bohg, Ken Goldberg, Abhinav Gupta, Abhishek Gupta, Dinesh Jayaraman, Joseph J Lim, Jitendra Malik, Roberto Martín-Martín, Subramanian Ramamoorthy, Dorsa Sadigh, Shuran Song, Jiajun Wu, Michael C. Yip, Yuke Zhu, Thomas Kollar, Sergey Levine, and Chelsea Finn. Droid: A large-scale in-the-wild robot manipulation dataset, 2025.
- [19] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024. 2
- [20] Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success, 2025.
- [21] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020. 8
- [22] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration. In *MLSys*, 2024. 5
- [23] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *arXiv preprint arXiv:2306.03310*, 2023. 4, 6, 1

- [24] Yunchao Ma, Yizhuang Zhou, Yunhuan Yang, Tiancai Wang, and Haoqiang Fan. Running vlas at real-time speed. *arXiv preprint arXiv:2510.26742*, 2025. 1, 2
- [25] Michael Matthews, Michael Beukman, Chris Lu, and Jakob Foerster. Kinetix: Investigating the training of general agents through open-ended physics-based control tasks. 2025. 2, 6
- [26] NVIDIA, Nikita Cherniadev, Johan Bjorck, and Fernando Castañeda, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llongtop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzheng Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An open foundation model for generalist humanoid robots. In *ArXiv Preprint*, 2025. 1, 2
- [27] 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, 2025.
- [28] Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Yan Ding, Zhigang Wang, JiaYuan Gu, Bin Zhao, Dong Wang, and Xuelong Li. Spatialvla: Exploring spatial representations for visual-language-action model, 2025.
- [29] Kohei Sendai, Maxime Alvarez, Tatsuya Matsushima, Yutaka Matsuo, and Yusuke Iwasawa. Leave no observation behind: Real-time correction for vla action chunks. *arXiv preprint arXiv:2509.23224*, 2025. 1, 2, 3
- [30] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision, 2024.
- [31] Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, et al. Smovla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025. 1, 2, 3, 6, 7
- [32] Zhi Su, Bike Zhang, Nima Rahamanian, Yuman Gao, Qiayuan Liao, Caitlin Regan, Koushil Sreenath, and S Shankar Sastry. Hitter: A humanoid table tennis robot via hierarchical planning and learning. *arXiv preprint arXiv:2508.21043*, 2025.
- [33] Galaxeal Team. Galaxeal g0: Open-world dataset and dual-system vla model. *arXiv preprint arXiv:2509.00576v1*, 2025.
- [34] Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, et al. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025. 1
- [35] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34:24261–24272, 2021. 6
- [36] Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, Yixin Peng, Feifei Feng, and Jian Tang. Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation, 2025.
- [37] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models, 2024. 5
- [38] Zhaoshu Yu, Bo Wang, Pengpeng Zeng, Haonan Zhang, Ji Zhang, Lianli Gao, Jingkuan Song, Nicu Sebe, and Heng Tao Shen. A survey on efficient vision-language-action models, 2025.
- [39] Wenyao Zhang, Hongsi Liu, Zekun Qi, Yunnan Wang, Xinqiang Yu, Jiazhao Zhang, Runpei Dong, Jiawei He, Fan Lu, He Wang, Zhizheng Zhang, Li Yi, Wenjun Zeng, and Xin Jin. Dreamvla: A vision-language-action model dreamed with comprehensive world knowledge, 2025.
- [40] Qingqing Zhao, Yao Lu, Moo Jin Kim, Zipeng Fu, Zhuoyang Zhang, Yecheng Wu, Zhaoshuo Li, Qianli Ma, Song Han, Chelsea Finn, Ankur Handa, Ming-Yu Liu, Donglai Xi-ang, Gordon Wetzstein, and Tsung-Yi Lin. Cot-vla: Visual chain-of-thought reasoning for vision-language-action models, 2025.
- [41] 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. 1
- [42] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware, 2023. 3
- [43] 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*, pages 2165–2183. PMLR, 2023. 2

VLASH: Real-Time VLAs via Future-State-Aware Asynchronous Inference

Supplementary Material

7. Appendix

7.1. SmoVLA Results on LIBERO Benchmarks

To further evaluate the generalization of VLASH across different VLAs, we conduct additional experiments on SmoVLA-450M [31], a compact yet efficient vision-language-action model. Following the same experimental setup as described in Sec. 5.1.2, we fine-tune SmoVLA on the LIBERO benchmark [23] for 30K iterations with a batch size of 32. We evaluate the model across four LIBERO sub-benchmarks (Spatial, Object, Goal, and LIBERO-10) under various inference delays ranging from 0 to 4 steps, with an execution horizon of $K = 5$.

As shown in Table 4, VLASH achieves consistent speedups across all inference delays when applied to SmoVLA. At delay 2 and 3, VLASH achieves up to $1.35\times$ speedup compared to synchronous inference. While the success rate shows minor variations across different delays, VLASH at delay 3 achieves 79.06% success rate, which is comparable to the synchronous baseline (78.96%), demonstrating that VLASH can maintain performance while providing significant latency improvements. These results further validate that VLASH generalizes effectively across different VLA architectures.

7.2. Experimental Details

We present the detailed training hyperparameters used for fine-tuning VLAs in our experiments in Table 5. For all experiments on LIBERO benchmarks and real-world tasks, we use same hyperparameters to ensure fair comparison across different methods and models. These hyperparameters are carefully tuned to balance training stability and convergence speed while preventing overfitting on the downstream tasks.

Table 5. Training hyperparameters for fine-tuning VLAs. We use these hyperparameters for fine-tuning $\pi_{0.5}$ and SmoVLA on LIBERO and real-world tasks.

Hyperparameter	Value
<i>Training Configuration</i>	
Batch Size	32
Training Steps	30,000
<i>Optimizer (AdamW)</i>	
Learning Rate	5e-5
Betas	[0.9, 0.95]
Weight Decay	1e-10
<i>Learning Rate Scheduler</i>	
Type	Cosine Decay with Warmup
Warmup Steps	1,000
Peak Learning Rate	5e-5
Decay Learning Rate	2.5e-6
Decay Steps	30,000

7.3. Supplementary Demo Video

We provide comprehensive video demonstrations comparing our method against synchronous and naive asynchronous baselines across various real-world manipulation tasks. All demonstrations are conducted using $\pi_{0.5}$ [16] deployed on a laptop with NVIDIA RTX 5090 GPU, achieving an inference frequency of 15Hz.

We showcase the following tasks in the supplementary materials:

- **Ping-pong:** Interactive rallies with a human player, demonstrating rapid reaction capabilities.
- **Whack-a-mole:** Fast-response game requiring quick detection and precise striking motions.
- **Pick and place:** Standard manipulation task showing smooth motion control.

Table 4. Performance on LIBERO benchmarks with SmoVLA-450M and different inference delays. We evaluate SmoVLA across four LIBERO sub-benchmarks (Spatial, Object, Goal, LIBERO-10) under various inference delays (1 to 4 steps). **SR:** average success rate; **Steps:** average execution steps to task completion; **Latency:** inference latency in seconds.

Method	Delay	Success Rate (%)				Average			Improvement	
		Spatial	Object	Goal	LIBERO-10	SR	Steps	Latency (s)	Δ SR	Speedup
Sync	0	81.25	92.91	85.83	55.83	78.96	198.70	8.82	-	-
VLASH (Async)	1	80.00	92.91	82.29	53.13	77.08	199.08	7.53	-1.88	1.17\times
	2	78.54	92.08	86.88	55.00	78.12	197.83	6.53	-0.84	1.35\times
	3	79.79	94.17	87.50	54.79	79.06	197.68	6.52	+0.10	1.35\times
	4	73.13	93.54	84.38	53.33	76.09	203.64	6.72	-2.87	1.31\times

- **Folding clothes:** Complex manipulation requiring coordinated movements.

We compare three inference modes: synchronous inference, naive asynchronous inference, and VLASH. Additionally, we demonstrate the effects of action quantization, showing how our method can achieve further speedups while maintaining task performance.

The video demonstrations clearly show that VLASH produces noticeably smoother motions and faster task completion compared to both synchronous and naive asynchronous baselines. The synchronous baseline often exhibits stuttering behavior due to action stalls, while naive asynchronous inference suffers from prediction-execution misalignment that leads to erratic movements. In contrast, VLASH maintains fluid motion throughout task execution while achieving significant speedup. We encourage readers to view the video to appreciate the dynamic performance improvements of our approach.

7.4. Architectural Modifications

A key advantage of VLASH is that it requires *no architectural modifications* to achieve effective performance across diverse VLA models. Since all current VLA models accept robot state inputs, VLASH can be applied directly by simply offsetting the state information during fine-tuning to account for inference delay. This straightforward approach enables the model to learn the temporal alignment between delayed observations and corresponding actions without any changes to the model architecture.

For standard VLA architectures like π_0 [3] and SmoVLA [31], which incorporate a state projection layer to embed proprioceptive state vectors into continuous representations before feeding them into the transformer backbone, VLASH integrates seamlessly and achieves excellent results out of the box.

We further note that VLASH also works directly with $\pi_{0.5}$ [16] without modifications, as demonstrated in our experiments in Table 1. However, $\pi_{0.5}$ employs a unique design that converts numerical state values into text tokens and appends them to the language prompt. This text-based encoding forces numerical state values through tokenization and one-hot encoding, disrupting their inherent numerical structure and making it more challenging for the model to learn from state information. For such architectures, we find that adding a lightweight state projection like the design of π_0 and injecting the resulting embeddings back into their original positions can further enhance smoothness and stability. A simpler alternative is to incorporate the projected state embeddings into the AdaRMSNorm layers as conditioning signals alongside timestep embeddings. While entirely optional (and VLASH already performs well without it), this small architectural enhancement consistently improves control smoothness for $\pi_{0.5}$. Importantly, the ad-

ditional parameters introduced by this state projection layer are negligible: it consists only of a linear mapping from the state dimension to the hidden dimension. Moreover, because it is zero-initialized, it completely preserves the pretrained model’s performance during the initial stages of fine-tuning.