Chunk-Level Routing for Vision-Language-Action Models

Tian Xie
xie00529@umn.edu
University of Minnesota
Minneapolis, MN, USA

Youngjin Hong
hong0745@umn.edu
University of Minnesota
Minneapolis, MN, USA

Zishen Wan
Georgia Institute of Technology
Atlanta, GA, USA

Yang (Katie) Zhao
yangkatiezhao@umn.edu
University of Minnesota
Minneapolis, MN, USA

Abstract

Vision-Language-Action (VLA) models have achieved strong performance on robotic manipulation tasks, yet existing evaluation protocols often exclude inference latency from simulated-time execution and rely primarily on binary task success. We introduce **LIBERO-LQ**, a **Latency and Quality-aware** benchmark that advances the environment during policy inference and evaluates synchronous and asynchronous execution under matched latency. Beyond task success, LIBERO-LQ measures trajectory-level quality, including smoothness, stability, motion efficiency, and estimated actuator energy. Evaluations on LIBERO-LQ reveal a continuity–reactivity trade-off in fixed real-time chunking strategies: retaining prior continuation improves motion continuity, but can constrain corrective replanning when observations change. To address this trade-off, we introduce **Adaptive Continuation Routing (ACR)**, a lightweight router that adaptively retains or releases continuation at each replanning step. Extensive experiments show that ACR maintains high task success while preserving trajectory quality under inference latency.

Keywords: Vision-Language-Action Models, Embodied AI

1 Introduction

Vision-Language-Action (VLA) models have recently emerged as a powerful paradigm for general-purpose robotic manipulation. Representative systems include RT-1, RT-2, OpenVLA, and π_0 [3, 6, 13, 30]. These models are commonly evaluated on standardized simulation benchmarks, including LIBERO, CALVIN, ManiSkill2, BEHAVIOR-1K, and EmbodiedBench [10, 14, 18, 20, 29].

Recent studies have shown that inference latency can substantially degrade VLA performance during deployment. Delay-Aware Diffusion Policy [17] demonstrates a monotonic decrease in task success as execution latency increases due to observation–execution mismatch. A2C2 [23] and AsyncVLA [11] further show that delayed and asynchronous execution require real-time correction mechanisms to maintain performance. These results establish inference latency as a first-order factor in closed-loop robot behavior.

However, widely used robotic manipulation benchmarks, such as LIBERO [18], typically do not represent policy inference latency in simulated-time execution: although inference consumes wall-clock time, the environment remains paused until a new action is available. As a result, policies are evaluated under effectively zero inference latency, obscuring latency-induced degradation in closed-loop execution.

A second limitation is that existing evaluation primarily relies on binary task success. While success rate directly measures task completion, it provides only a coarse view of execution behavior. Two VLA systems with similar success rates may produce substantially different trajectories in terms of smoothness, stability, motion efficiency, and actuation effort. These differences are not captured by success-only evaluation, yet they are critical for deploying VLA policies on physical robots.

To address this gap, we propose a latency-aware benchmark for Vision-Language-Action models that explicitly decouples policy inference from control execution and models inference latency as a first-class experimental variable. By incorporating realistic execution semantics and evaluating policies beyond success rates alone, our benchmark provides a more faithful and fair assessment of VLA systems under realistic deployment conditions. **Our contributions are as follows**:

- We introduce **LIBERO-LQ**, a latency- and quality-aware benchmark for VLA policies. In addition to task success and completion time, LIBERO-LQ reports trajectory-level execution-quality metrics, including jerk, end-effector oscillation, trajectory and joint-rotation efficiency.
- Using LIBERO-LQ, we re-evaluate representative VLA models and execution policies under latency-aware execution, showing that success-only evaluation obscures substantial differences in closed-loop behavior and trajectory quality. In particular, we reveal a continuity–reactivity trade-off in continuation-conditioned real-time chunking: retaining the continuation reference improves motion smoothness, but can hinder corrective replanning when the current observation calls for different future actions.

- To navigate this trade-off, we propose **ACR**, a lightweight chunk-level router that selects, at each replanning step, between prefix-conditioned generation and free generation. The router is compatible with real-time chunking methods such as RTC and Legato, and adds negligible inference overhead.

2 Background

Vision-Language-Action (VLA) models [6, 13, 16, 30] map visual observations and natural language instructions directly to robot actions using a learned policy. Recent systems have demonstrated strong task generalization by leveraging Vision-Language Model (VLM) backbones [1, 7, 15, 24]. By combining pretrained visual encoders [21] with large language models [26, 27] and training on large-scale robot demonstration datasets [8, 9, 12, 22, 28], VLA policies have emerged as generalist controllers capable of executing diverse manipulation tasks. As these models scale in size and capability, inference latency has become a central deployment challenge, fundamentally shaping how policy inference is coupled with action execution. Two representative architectural paradigms are OpenVLA [13], which autoregressively predicts discretized action tokens, and $\pi_{0.5}$ [2], which incorporates a flow-matching action expert to generate continuous action chunks.

2.1 Execution Policy

Early VLA systems commonly employ **synchronous execution**, where the robot must wait for policy inference before receiving new actions, leading to blocking behavior under non-negligible latency [6, 13, 30]. **Asynchronous execution** avoids such stalls by launching inference before the current action chunk is exhausted, allowing prediction and execution to overlap. Its simplest form, termed **Naive Async**, immediately switches to the newly predicted chunk once it becomes available [4]. However, because the new chunk is generated without explicitly accounting for the pending actions executed during inference, it can be inconsistent with the ongoing trajectory, causing discontinuous or jerky motion at chunk boundaries.

To address this issue, Physical Intelligence proposed Real-Time Chunking (RTC) [4], an inference-time method that formulates asynchronous chunk generation as an inpainting problem: actions committed during inference are frozen as a prefix, while the remaining actions are generated under guidance from the previous chunk to improve inter-chunk continuity. Since inference-time inpainting introduces additional denoising overhead, training-time RTC [5] simulates inference delays during training and directly conditions the policy on committed action prefixes, eliminating this additional deployment-time cost. Complementarily, VLASH [25] addresses prediction–execution misalignment by estimating the execution-time robot state from previously issued actions, enabling future-state-aware asynchronous prediction without additional inference overhead. More recently, Legato [19] targets the lack of policy-native continuation in RTC-style methods by learning continuation behavior through schedule-shaped conditioning and reshaped flow dynamics, reducing spurious multimodal switching across chunks.

Table 1. Execution policies evaluated in LIBERO-LQ. Bold policies rely on flow-matching action generation and are not applied to autoregressive action decoding in our evaluation.

Policy	Action During Inference	Latency-Handling Mechanism
Sync	Wait / hold action	Applies a newly predicted chunk only after inference completes, modeling blocking behavior.
Naive Async	Remaining previous-chunk actions	Directly switches to the new chunk after inference, without cross-chunk alignment.
RTC-Zero	Remaining previous-chunk actions	Freezes the committed prefix and generates the remaining actions without soft overlap guidance.
RTC-Exp	Remaining previous-chunk actions	Freezes the committed prefix and applies exponentially decayed guidance over the overlapping region.
Training-time RTC*	Remaining previous-chunk actions	Predicts the continuation conditioned on committed prefixes learned under randomized training-time delays.
VLASH*	Remaining previous-chunk actions	Predicts from a rolled-forward execution-time robot state to reduce prediction–execution misalignment.
Legato*	Remaining previous-chunk actions	Learns continuation-aware flow dynamics for smooth asynchronous chunk generation.

* Requires method-specific fine-tuning; policies without this marker are applied at inference time without retraining the underlying VLA checkpoint. **Bold** denotes policies applicable only to flow-matching VLA models.

3 Libero-LQ

3.1 VLA Models and Execution Policies

LIBERO-LQ evaluates two factors: the VLA model that predicts actions and the execution policy that determines robot behavior while a new prediction is unavailable due to inference latency. This distinction matters because systems with similar task success under zero-latency evaluation may behave differently once inference delay is represented in simulated time. For methods requiring dedicated conditioning or training, we evaluate the corresponding model-policy configuration rather than treating the two factors as fully interchangeable.

Let Δt_c denote the control timestep, and let $\Delta t_{\text{inf}}^{\text{sim}}$ denote the inference delay represented in simulated time, obtained either from measured wall-clock latency or from a controlled latency sweep. The corresponding number of elapsed control steps is $d = \lceil \Delta t_{\text{inf}}^{\text{sim}} / \Delta t_c \rceil$. At policy query step k, the VLA model produces an action chunk of horizon H, denoted as $\boldsymbol{A}_k = [\boldsymbol{a}_k^{(0)}, \boldsymbol{a}_k^{(1)}, \ldots, \boldsymbol{a}_k^{(H-1)}]$.

During the subsequent d control steps, the new chunk is unavailable while the simulator continues to evolve. The execution policy specifies both the actions applied during this interval and how the returned chunk is connected to the ongoing trajectory. A synchronous policy executes a

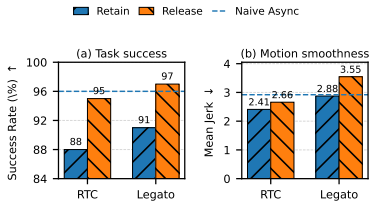

Figure 1. Fixed continuation decisions reveal a success–smoothness trade-off on LIBERO-10. Dashed lines indicate Naive Async.

prescribed waiting or holding behavior until inference completes, whereas an asynchronous policy continues executing available actions from the previous chunk. Continuation-based methods additionally specify how the unexecuted remainder of the previous chunk guides the new prediction. Table 1 summarizes the execution policies evaluated in LIBERO-LQ.

3.2 Evaluation Metrics

Existing robotic manipulation benchmarks primarily evaluate policies using functional metrics such as binary task success. While effective for measuring whether a task is completed, such metrics provide limited insight into how the task is executed or how inference latency affects deployment-relevant behavior. LIBERO-LQ therefore reports a set of non-functional metrics. Details are shown in Appendix A.

4 Adaptive Continuation Routing (ACR)

4.1 Problem Formulation: Non-Stationary Reliability of Continuation Conditioning

We consider continuation-conditioned real-time action chunking methods that generate a new action chunk while part of the previously predicted chunk is still being executed. Let $\boldsymbol{A}_k = \left[\boldsymbol{a}_{k,0}, \ldots, \boldsymbol{a}_{k,H-1}\right]$ denote the action chunk produced at replanning step k, where H is the chunk horizon, s is the execution horizon, and d is the inference delay measured in control steps. After the first s actions of the previous chunk have been executed, its remaining tail is $\bar{\boldsymbol{A}}_{k-1} = \boldsymbol{A}_{k-1}[s : H]$, which contains $H - s$ actions aligned with the first $H - s$ positions of the newly generated chunk.

Real-time continuation methods such as RTC and Legato use $\bar{\boldsymbol{A}}_{k-1}$ to encourage consistency between consecutive action chunks. RTC realizes this behavior through inference-time guidance over the overlapping actions, whereas Legato controls the continuation effect through its ramp-length parameter. Although the mechanisms differ, both methods expose a continuation strength that can be retained or released at each replanning boundary.

The overlap between the previous tail and the new chunk contains two semantically different regions. During the d control steps required for inference, the first d actions of $\bar{\boldsymbol{A}}_{k-1}$ are executed before the newly generated chunk becomes available. These actions therefore form a *committed prefix*,

$$\boldsymbol{P}_{k-1} = \bar{\boldsymbol{A}}_{k-1}[0 : d], \tag{1}$$

which must be preserved in the new chunk:

$$\boldsymbol{A}_k[0 : d] = \boldsymbol{P}_{k-1}. \tag{2}$$

The remaining aligned actions,

$$\boldsymbol{S}_{k-1} = \bar{\boldsymbol{A}}_{k-1}[d : H - s], \tag{3}$$

form an *editable overlap*. These actions have not yet been executed when the new prediction becomes available, so continuation over this region can be preserved to encourage smooth chunk transitions or released to allow corrective replanning. The final s actions of the new chunk have no aligned counterpart in $\bar{\boldsymbol{A}}_{k-1}$ and therefore form an unconstrained future region.

Existing continuation-conditioned methods typically use a fixed continuation configuration throughout an episode, although the value of the previous tail varies across execution stages. Retaining continuation can smooth transport motions, but after contacts, phase transitions, or large observation changes, the stale tail may hinder corrective replanning. This motivates an adaptive router that decides whether to retain or release continuation at each replanning boundary.

Figure 1 shows this success–smoothness trade-off. For RTC, Release improves success from 88% to 95%, while mean jerk increases from 2.41 to 2.66. For Legato, success rises from 91% to 97%, but jerk increases from 2.88 to 3.55. These results suggest that continuation should be selected adaptively rather than fixed throughout an episode.

We formulate the routing decision as $b_k \in \{\textsc{Release}, \textsc{Retain}\}$, applied only to the editable overlap $\boldsymbol{S}_{k-1}$; the committed prefix $\boldsymbol{P}_{k-1}$ is preserved under both decisions.

Routed continuation. The routing decision controls whether soft continuation is applied over the editable overlap. Under Retain, the previous action tail provides a soft reference for the editable region; under Release, this soft reference is removed while the committed prefix remains unchanged:

$$C_k = \begin{cases} \boldsymbol{S}_{k-1}, & b_k = \textsc{Retain}, \\ \varnothing, & b_k = \textsc{Release}, \end{cases} \tag{4}$$

where C_k denotes the soft continuation context used for generating the next action chunk. In RTC, Release is implemented by setting the soft guidance weights in the editable overlap to zero. In Legato, it is implemented by setting the ramp length to $r_k = 0$ for the current replan.

4.2 ACR: Learned Release-or-Retain Decision

Router formulation. We train a separate lightweight binary router for each underlying VLA model and omit the model index when it is clear from context. At each replanning boundary, the router predicts the probability of selecting Release as $p_k = \mathrm{ACR}(\boldsymbol{z}_k, \boldsymbol{R}_k)$, where $p_k \in [0, 1]$. The routing

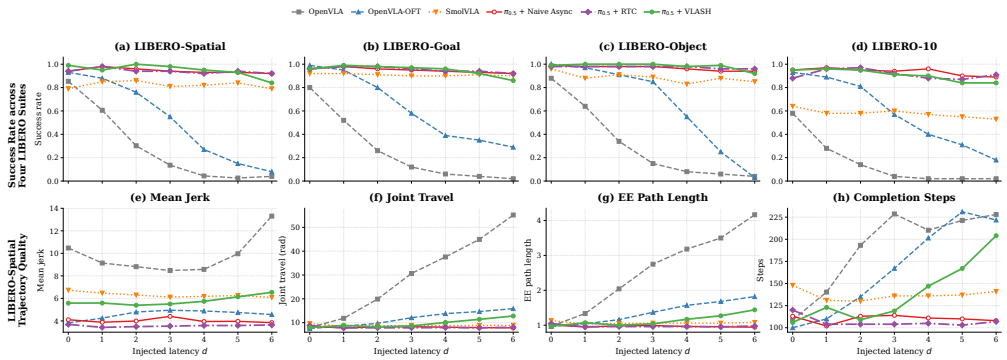

Figure 2. Latency-aware evaluation on LIBERO. Top: success rates under increasing inference latency d across four LIBERO suites. Bottom: trajectory-quality metrics on LIBERO-Spatial.

decision is b_k = RELEASE if $p_k > 0.5$, and b_k = RETAIN otherwise.

Here, $z_k \in \mathbb{R}^{D_z}$ denotes the mean-pooled VLM output produced from the current visual observation and language instruction at replanning step k, where D_z depends on the underlying VLA model. The reference $\boldsymbol{R}_k \in \mathbb{R}^{H \times D_a}$ is constructed from the remaining tail of the previously predicted action chunk and zero-padded to length H when necessary, where D_a is the action dimension. Training details are introduced in the supplementary material.

5 Experiments

5.1 Experimental Setup

We evaluate four representative VLA models: OpenVLA, OpenVLA-OFT, $\pi_{0.5}$, and SmolVLA. These models cover both single-action and action-chunking architectures, as well as synchronous and asynchronous execution policies. OpenVLA is an autoregressive VLA that predicts and executes a single action at each policy query. OpenVLA-OFT extends OpenVLA with action chunking and serves as a synchronous chunk-execution baseline, where the robot waits for inference to complete before applying a newly predicted chunk. In contrast, $\pi_{0.5}$ and SmolVLA natively support asynchronous action-chunk execution, allowing previously available actions to continue executing while a new chunk is being generated. SmolVLA further provides a lightweight real-time baseline with lower model complexity. All models are implemented using their official repositories and checkpoints.

5.2 Experiment Results

As shown in Fig. 2, synchronous baselines degrade sharply as latency increases, whereas asynchronous execution policies preserve substantially higher task success. Moreover, asynchronously executed VLAs exhibit better trajectory quality than their synchronous counterparts. In particular, RTC substantially reduces jerk, indicating smoother execution, but achieves a lower success rate than the $\pi_{0.5}$ naive asynchronous baseline on the LIBERO-10 suite. In contrast, integrating VLASH with $\pi_{0.5}$ leads to degraded trajectory quality despite maintaining relatively high task success.

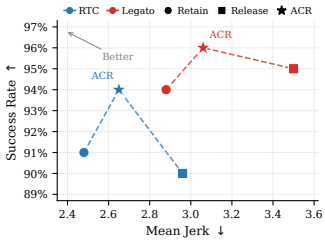

Figure 3. Success–smoothness comparison of fixed continuation strategies and ACR on LIBERO-10 under $d = 4$ and soft overlap length 6. ACR achieves the highest success rate for both RTC and Legato while avoiding the large jerk increase associated with always releasing continuation guidance.

5.3 ACR: Closing the Instance-Level Branch Gap

We now evaluate whether the learned router ACR can improve real-time execution by selecting between releasing and retaining editable continuation guidance at each replanning step. To demonstrate that ACR is generalizable across different continuation mechanisms, we instantiate it on two representative methods: RTC, an inference-time continuation smoothing strategy, and Legato, a fine-tuned continuation-aware policy. We train an independent ACR router for each instantiation and present the corresponding results in this section.

Figure 3 evaluates ACR on RTC and Legato under $d = 4$ and a soft overlap length of 6. For RTC, RETAIN is stronger than RELEASE among the two fixed strategies in both success rate and smoothness; nevertheless, ACR further improves success from 91% to 94% while keeping jerk substantially below that of RELEASE. For Legato, RELEASE slightly improves success over RETAIN, but incurs a large increase in jerk. ACR further raises success to 96% while reducing jerk from 3.50 to 3.06. These results show that ACR improves task success across distinct continuation mechanisms without inheriting the full smoothness cost of always releasing continuation guidance. More Ablation Experiments are included in supplementary materials

References

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

[2] Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y Galliker, et al. 2025. $\pi_0.5$: A Vision-Language-Action Model with Open-World Generalization. In *9th Annual Conference on Robot Learning*.

[3] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. 2024. π_0: A Vision-Language-Action Flow Model for General Robot Control. *arXiv preprint arXiv:2410.24164* (2024). https://www.pi.website/blog/pi0

[4] Kevin Black, Manuel Y Galliker, and Sergey Levine. 2025. Real-time execution of action chunking flow policies. *arXiv preprint arXiv:2506.07339* (2025).

[5] Kevin Black, Allen Z Ren, Michael Equi, and Sergey Levine. 2025. Training-time action conditioning for efficient real-time chunking. *arXiv preprint arXiv:2512.05964* (2025).

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

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

[8] Frederik Ebert, Yanlai Yang, Karl Schmeckpeper, Bernadette Bucher, Georgios Georgakis, Kostas Daniilidis, Chelsea Finn, and Sergey Levine. 2021. Bridge data: Boosting generalization of robotic skills with cross-domain datasets. *arXiv preprint arXiv:2109.13396* (2021).

[9] Hao-Shu Fang, Hongjie Fang, Zhenyu Tang, Jirong Liu, Chenxi Wang, Junbo Wang, Haoyi Zhu, and Cewu Lu. 2023. Rh20t: A comprehensive robotic dataset for learning diverse skills in one-shot. *arXiv preprint arXiv:2307.00595* (2023).

[10] Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, et al. 2023. Maniskill2: A unified benchmark for generalizable manipulation skills. *arXiv preprint arXiv:2302.04659* (2023).

[11] Yuhua Jiang, Shuang Cheng, Yan Ding, Feifei Gao, and Biqing Qi. 2025. AsyncVLA: Asynchronous Flow Matching for Vision-Language-Action Models. *arXiv preprint arXiv:2511.14148* (2025).

[12] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. 2024. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945* (2024).

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

[14] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, et al. 2023. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*. PMLR, 80–93.

[15] Feng Li, Renrui Zhang, Hao Zhang, Yuanhan Zhang, Bo Li, Wei Li, Zejun Ma, and Chunyuan Li. 2024. Llava-next-interleave: Tackling multi-image, video, and 3d in large multimodal models. *arXiv preprint arXiv:2407.07895* (2024).

[16] Xiang Li, Cristina Mata, Jongwoo Park, Kumara Kahatapitiya, Yoo Sung Jang, Jinghuan Shang, Kanchana Ranasinghe, Ryan Burgert, Mu Cai, Yong Jae Lee, et al. 2024. Llara: Supercharging robot learning data for vision-language policy. *arXiv preprint arXiv:2406.20095* (2024).

[17] Aileen Liao, Dong-Ki Kim, Max Olan Smith, Ali-akbar Aghamohammadi, and Shayegan Omidshafiei. 2025. Delay-Aware Diffusion Policy: Bridging the Observation-Execution Gap in Dynamic Tasks. *arXiv preprint arXiv:2512.07697* (2025).

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

[19] Yufeng Liu, Hang Yu, Juntu Zhao, Bocheng Li, Di Zhang, Mingzhu Li, Wenxuan Wu, Yingdong Hu, Junyuan Xie, Junliang Guo, et al. 2026. Learning native continuation for action chunking flow policies. *arXiv preprint arXiv:2602.12978* (2026).

[20] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. 2022. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters* 7, 3 (2022), 7327–7334.

[21] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. 2023. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193* (2023).

[22] Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. 2024. 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)*. IEEE, 6892–6903.

[23] Kohei Sendai, Maxime Alvarez, Tatsuya Matsushima, Yutaka Matsuo, and Yusuke Iwasawa. 2025. Leave no observation behind: Real-time correction for vla action chunks. *arXiv preprint arXiv:2509.23224* (2025).

[24] Andreas Steiner, André Susano Pinto, Michael Tschannen, Daniel Keysers, Xiao Wang, Yonatan Bitton, Alexey Gritsenko, Matthias Minderer, Anthony Sherbondy, Shangbang Long, et al. 2024. Paligemma 2: A family of versatile vlms for transfer. *arXiv preprint arXiv:2412.03555* (2024).

[25] Jiaming Tang, Yufei Sun, Yilong Zhao, Shang Yang, Yujun Lin, Zhuoyang Zhang, James Hou, Yao Lu, Zhijian Liu, and Song Han. 2025. Vlash: Real-time vlas via future-state-aware asynchronous inference. *arXiv preprint arXiv:2512.01031* (2025).

[26] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118* (2024).

[27] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[28] 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. 2023. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning*. PMLR, 1723–1736.

[29] Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, et al. 2025. Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. *arXiv preprint arXiv:2502.09560* (2025).

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

A Metrics

A.1 Inference Latency.

We distinguish between **real inference latency** $\Delta t_{\text{inf}}^{\text{real}}$, which measures the wall-clock time required for model inference on a given observation, and **simulated inference latency** $\Delta t_{\text{inf}}^{\text{sim}}$, which is an explicitly injected delay during simulation. The simulated latency is treated as an experimental variable and determines how long high-level actions are held while the simulator continues to advance.

A.2 Jerk.

To quantify trajectory smoothness, we measure *jerk*, defined as the third time derivative of the end-effector position, which captures abrupt changes in acceleration and is a standard indicator of motion smoothness in robotic manipulation. Let $\mathbf{p}_t \in \mathbb{R}^3$ denote the end-effector position at timestep t, sampled at a uniform interval Δt. We approximate the jerk vector using a third-order finite difference:

$$\mathbf{j}_t \approx \frac{\mathbf{p}_t - 3\mathbf{p}_{t-1} + 3\mathbf{p}_{t-2} - \mathbf{p}_{t-3}}{\Delta t^3}. \tag{5}$$

Trajectory smoothness is then quantified by the average jerk norm over the rollout:

$$J = \frac{1}{T-3} \sum_{t=4}^{T} \|\mathbf{j}_t\|_2, \tag{6}$$

where T denotes the trajectory length. Lower jerk norm values correspond to smoother and more stable motion, while higher values indicate abrupt or oscillatory behavior. [**?**]

A.3 Stability (End-Effector Oscillation).

While prior work measures execution stability using jerk-based trajectory instability in joint space, we instead focus on execution-level oscillations in task space, which are particularly prominent under inference latency.

Specifically, we measure stability by the average deviation of the end-effector position from its mean over a short temporal window of the last K steps:

$$\bar{\mathbf{p}} = \frac{1}{K} \sum_{i=T-K+1}^{T} \mathbf{p}_i, \quad \text{EE-Osc} = \frac{1}{K} \sum_{i=T-K+1}^{T} \|\mathbf{p}_i - \bar{\mathbf{p}}\|_2$$

where $\mathbf{p}_i \in \mathbb{R}^3$ denotes the end-effector position at timestep i. Higher EE-Osc indicates increased oscillatory behavior during task execution, often arising from latency-induced control inconsistencies.

A.4 Trajectory Efficiency.

We assess trajectory efficiency by the total trajectory length required to successfully complete the task. Since the minimal feasible trajectory length depends on task geometry and goal configuration, efficiency is evaluated via relative, within-task comparisons among successful trials.