RELATIVE TEMPORAL ALIGNMENT FOR REAL-TIME EXECUTION OF VISION-LANGUAGE-ACTION POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-Language-Action (VLA) policies commonly leverage action chunking to amortize the high computational cost of model inference. Action chunking reduces amortized computation but also lowers the rate at which fresh observations enter the policy. Synchronous execution may pause the robot between chunks, whereas asynchronous execution maintains continuous control but hands over a chunk conditioned on an earlier observation. Existing approaches address this challenge through chunk-boundary smoothing, future robot-state prediction, or stepwise visual feedback, yet they do not explicitly relate robot–environment changes during asynchronous inference to action residuals (i.e., corrections to the pending action sequence), limiting low-cost temporal alignment of pending actions. We propose Relative Temporal Alignment (RTA), a lightweight correction method decoupled from the base VLA policy. RTA keeps the base policy frozen and uses relative robot–environment changes during inference to efficiently correct each incoming action chunk before execution. This aligns actions conditioned on the observation at inference onset with the robot–environment state available when inference completes. Experiments across static simulation, dynamic simulation, and real-robot platforms show that RTA maintains robust performance under varying inference delays and improves task completion rates while incurring minimal correction overhead.

1 INTRODUCTION

Vision-Language-Action (VLA) policies integrate visual perception, language instructions, and robot control within a unified model, showing considerable promise for multitask manipulation and generalization across embodiments and environments (Ma et al., 2026; Brohan et al., 2022; 2023; Vuong et al., 2023; Team et al., 2024; Kim et al., 2024; Black et al., 2024; Intelligence et al., 2025; Shukor et al., 2025). These advances, however, come with increasingly substantial inference costs, i.e., large-scale perception backbones and sophisticated action-generation heads often cannot complete a forward pass within a single control cycle. Consequently, modern VLA policies commonly adopt action chunking, predicting a sequence of future actions in one inference call to amortize the fixed costs of visual encoding, language modeling, and diffusion- or flow-matching-based sampling (Zhao et al., 2023; Chi et al., 2025; Lipman et al., 2022; Pertsch et al., 2025).

However, action chunking only reduces the average computation per control step, but it does not shorten the wall-clock time required for the base policy to produce a new chunk. The execution framework may therefore need to strike a balance between continuous control and timely action updates. Specifically, synchronous inference pauses the robot while waiting for each new chunk, whereas asynchronous inference continues to execute the previous chunk. Although asynchronous execution avoids idle time, the incoming chunk remains conditioned on the observation captured at the start of inference. If inference begins at control step t and returns a chunk d steps later at $t + d$, both the robot and its environment may have changed in the interim. Observation staleness is thus the primary source of degradation in asynchronous inference: the chunk about to take control is conditioned on a world state that is already d steps out of date (Agouzoul, 2026).

Prior works have addressed this challenge from several perspectives, but remain limited in effectiveness and efficiency. RTC and VLASH focus on chunk-boundary consistency and predictable robot

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

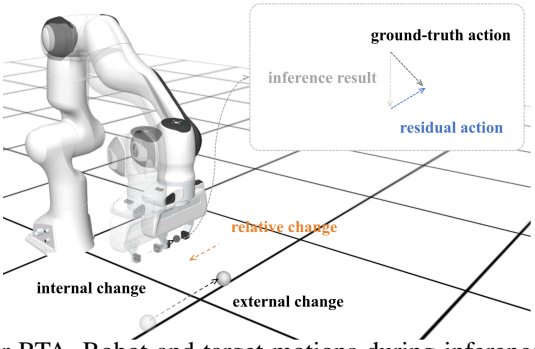


Figure 1: Motivation for RTA. Robot and target motions during inference alter their relative configuration. RTA predicts an action residual to align the stale action chunk with the configuration at handover.

motion, respectively (Black et al., 2026; Tang et al., 2025). Consequently, they struggle to account for misalignment caused by independent environmental changes, leaving pending actions insufficiently adapted to the altered robot–environment relationship. A2C2 incorporates real-time visual feedback (Sendai et al., 2025), but its relatively large correction model, stepwise invocation, and tight coupling to base-policy features incur substantial correction overhead and limit adaptability across policies. Overall, RTC, VLASH, and A2C2—the direct comparison methods in this work—do not explicitly model the accumulated change in the robot–environment relationship during inference as a direct signal for correcting pending actions.

Our key insight is that the action error induced by observation staleness depends on the relative motion between the robot and its environment during the inference window. Accordingly, the relative change over this interval is strongly correlated with the action correction required at execution time. During asynchronous inference, the robot moves under previously issued actions, while the target object and its surrounding environment may change independently of the robot. Figure 1 illustrates this intuition. At the start of inference, the object lies directly beneath the gripper, prompting the base policy to generate a downward motion. By the time inference finishes, a lateral displacement has developed between the object and the gripper, and the pending action should include a corresponding lateral residual. Therefore, capturing changes in the robot–environment relationship during inference provides a direct basis for predicting targeted corrections to stale actions.

Inspired by this insight, we propose Relative Temporal Alignment (RTA), which performs lightweight asynchronous temporal correction by leveraging relative robot–environment changes during inference. Specifically, RTA captures the accumulated relative change over each inference interval by combining low-resolution optical flow between global observations with the corresponding robot-state displacement. A lightweight dual-branch correction head translates these complementary cues into a one-shot action residual for the valid suffix of each incoming action chunk. By explicitly accounting for independent environmental changes, avoiding costly stepwise correction, and keeping the base VLA policy frozen, RTA addresses both the effectiveness and efficiency limitations of prior approaches. Across varying inference delays on the static LIBERO benchmark, RTA achieves task success rates comparable to A2C2, a representative stepwise correction method. In the dynamic DOMINO setting, RTA outperforms A2C2 and approaches ideal synchronous execution, while real-robot experiments further demonstrate its effectiveness. Each correction requires only 20.88 MFLOPs, with a P95 latency of 1.16 ms. Our contributions are summarized as follows:

- We identify a close connection between robot–environment changes during asynchronous inference and the action corrections they necessitate, offering a new modeling perspective on observation staleness.
- We introduce RTA, a framework decoupled from the base VLA policy that models relative change using images and robot states captured before and after inference. RTA maps this change to an action residual, temporally aligning actions generated from the observation at inference onset with the robot–environment state at inference completion.
- We conduct systematic evaluations on static and dynamic benchmarks as well as a real-robot platform, demonstrating RTA's effectiveness across varying inference delays and environmental changes.

2 Related Works

2.1 Vision-Language-Action Policies and Action Chunking

Vision-Language-Action (VLA) models jointly process visual observations, language instructions, and robot states to map high-level semantics to low-level control, enabling a unified policy to handle diverse tasks and embodiments (Ma et al., 2026; Brohan et al., 2022; 2023; Vuong et al., 2023; Team et al., 2024; Kim et al., 2024; Black et al., 2024; Intelligence et al., 2025). Compact models and client-server architectures further facilitate asynchronous deployment on resource-constrained platforms (Shukor et al., 2025).

ACT, Diffusion Policy, and flow-matching VLAs commonly use action chunking to predict short trajectories, improving temporal consistency while amortizing inference costs (Zhao et al., 2023; Chi et al., 2025; Lipman et al., 2022; Black et al., 2024). FAST instead represents high-frequency action sequences through frequency-domain compression, illustrating that action-sequence representation is itself a key design choice for VLA training and inference efficiency (Pertsch et al., 2025). However, action chunking reduces the frequency at which fresh observations are incorporated, making chunk handoff critical to real-time control. RTA leaves the base VLA unchanged and performs lightweight temporal alignment only before a new chunk takes control.

2.2 Asynchronous Execution of Action-Chunking Policies

Synchronous execution waits for each new chunk and may interrupt control. Asynchronous execution maintains continuity by consuming buffered actions during inference, but the new chunk consequently takes control from a stale observation. Existing methods address this temporal mismatch through constrained generation, future-state conditioning, online correction, or system-level co-design.

RTC constrains flow inpainting with the actions executed during inference, improving chunk continuity without retraining the base policy, but it applies only to diffusion or flow models and requires additional vector–Jacobian products at every sampling step (Black et al., 2026). Training-time RTC moves prefix conditioning into training, avoiding deployment-time backward passes at the cost of modifying the objective and generator interface (Black et al., 2025). VLASH instead rolls the robot state forward to the expected handoff and conditions generation on that future state, compensating primarily for predictable robot motion while requiring temporal-offset fine-tuning (Tang et al., 2025).

A2C2 predicts action residuals at every control step from fresh observations and base-policy features, enabling frequent feedback but incurring computational costs that grow with the execution horizon and requiring interfaces tied to the policy and embodiment (Sendai et al., 2025). DynamicVLA co-designs a compact model, continuous inference, action streaming, and dynamic data, and thus depends on a specialized model and data stack (Xie et al., 2026). In contrast, RTA freezes the base policy and performs a single correction before chunk handoff using images and robot states captured before and after inference, allowing it to serve as a standalone module for different action-chunking VLAs.

Recent work also improves delay robustness through training and post-training. REMAC learns masked action chunks and preserves their prefixes during sampling, addressing both inter-chunk discontinuity and intra-chunk inconsistency (Wang et al., 2026). DEFLECT constructs temporal counterfactual preference pairs from future execution-time observations and uses offline post-training to improve the delay robustness of asynchronous VLAs (Zhu et al., 2026).

Another family modifies test-time selection or action generation. RACE searches from the current robot state for a smoother, dynamically reachable candidate chunk (Kim et al., 2026). AR-VLA maintains a long action history with a persistent autoregressive action expert and temporally re-anchors asynchronously refreshed vision-language prefixes (Hu et al., 2026). Beyond VLAs, Reactive Diffusion Policy combines low-frequency latent action chunks with high-frequency tactile feedback, further demonstrating the feasibility of slow-fast correction for chunking policies (Xue et al., 2025).

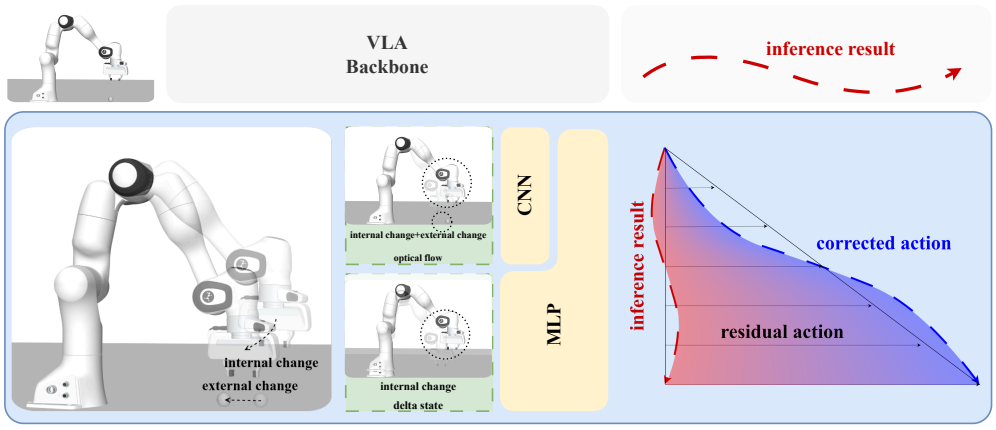


Figure 2: Overview of RTA. Optical flow and robot-state differences capture relative changes over the inference interval, enabling a lightweight correction head to correct the valid suffix of the asynchronously generated action chunk.

3 PRELIMINARIES

Let t index the controller timestep. At timestep t, o_t denotes the visual observation, I_t the global-camera image contained in o_t, $s_t \in \mathbb{R}^{D_s}$ the robot state, and l the language instruction. Conditioned on these inputs, the base action-chunking policy π predicts an action chunk A_t of length H:

$$A_t = [a_{0|t}, a_{1|t}, \ldots, a_{H-1|t}] \sim \pi(\cdot \mid o_t, s_t, l). \tag{1}$$

The parameter H is the *prediction horizon*, K ($\leq H$) is the *execution horizon*, and D_a is the action dimension. Let Δt denote the controller sampling period, and let δ denote the continuous inference latency—the wall-clock interval between acquiring an observation and making the corresponding action chunk available to the controller. We discretize this latency into a delay of $d = \lfloor \delta / \Delta t \rceil$ control steps, where $0 \leq d < H$.

Under this delay, a chunk requested at timestep t becomes available only at $t + d$. The execution times corresponding to indices $0, \ldots, d-1$ have therefore already elapsed. The asynchronous system discards these expired entries and transfers control to the valid suffix $A_t[d:] = [a_{d|t}, \ldots, a_{H-1|t}]$.

As inference proceeds, the robot continues to execute previously available actions while the environment may evolve independently. Thus, although A_t is conditioned on (o_t, s_t), it takes control under the updated conditions (o_{t+d}, s_{t+d}). We refer to this temporal mismatch as *prediction-execution misalignment*. RTA uses the relative changes accumulated over the inference interval to correct the valid suffix $A_t[d:]$ before it takes over execution.

4 RELATIVE TEMPORAL ALIGNMENT

Figure 2 provides an overview of RTA. At time step t, the system records I_t and s_t and invokes the base VLA asynchronously, while the robot continues to execute the current action chunk. Once the new chunk becomes available at $t + d$, the system acquires I_{t+d} and s_{t+d} and estimates the relative change between the robot and the environment over the inference interval. The correction head predicts $r_{t,d}$ in a single forward pass. RTA then corrects the valid suffix $A_t[d:]$ according to its action representation and atomically updates the action buffer. This procedure is completed before the new chunk takes control.

4.1 RELATIVE CHANGE MODELING

As illustrated in Figure 1, under asynchronous inference, the pending action chunk is conditioned on the robot–environment configuration at inference onset but takes control only after this configuration has evolved. The accumulated relative change over the inference interval therefore captures the discrepancy between the configuration assumed by the pending actions and that encountered at handover, which largely determines the direction and magnitude of the action residual required for correction. To correct actions predicted from stale observations, RTA models the relative change accumulated between observation acquisition and action takeover. Specifically, we decompose the

scene change over the inference interval into an internal component $C^{\text{int}}_{t,d}$ induced by the robot's motion and an external component $C^{\text{ext}}_{t,d}$ induced by the independent evolution of the environment. The relative change relevant to action correction is

$$C^{\text{rel}}_{t,d} = C^{\text{int}}_{t,d} - C^{\text{ext}}_{t,d} = 2C^{\text{int}}_{t,d} - \left(C^{\text{int}}_{t,d} + C^{\text{ext}}_{t,d}\right).$$

Because the external change is not directly observable, this identity reduces relative-change estimation to jointly modeling the internal change and the aggregate change induced by the robot and the environment.

The internal change is directly characterized by the difference between the robot states at the beginning and end of inference:

$$\Delta s_{t,d} = s_{t+d} - s_t. \tag{2}$$

Meanwhile, a global-view camera captures both robot motion and environmental evolution. We compute dense optical flow $F_{t,d} = \text{Flow}(I_t, I_{t+d})$ between the images acquired before and after inference to characterize the combined apparent motion associated with $C^{\text{int}}_{t,d} + C^{\text{ext}}_{t,d}$ in the image plane (Fortun et al., 2015; Teed & Deng, 2020).

Only the dominant direction and magnitude of motion over the inference interval are needed; the optical flow can therefore be computed at a substantially lower resolution than the visual input to the base VLA, keeping the correction overhead low. The state delta and optical flow provide complementary cues: the former directly measures the robot's motion, whereas the latter summarizes visual changes jointly induced by the robot and the environment. RTA encodes the two signals with separate branches and fuses them in a shared feature space to learn the relative-change representation and predict an action-space correction residual:

$$r_{t,d} = g_\theta(F_{t,d}, s_{t+d} - s_t) \in \mathbb{R}^{D_a}. \tag{3}$$

The corrector g_θ bypasses explicit reconstruction of the unobservable $C^{\text{ext}}_{t,d}$. It instead uses the state branch to identify robot-induced motion and the optical-flow branch to capture the composite scene change, learning a direct mapping from relative change to action correction. Preserving a base controller while learning an additive correction follows the broader residual-policy paradigm (Silver et al., 2018). The base policy π remains frozen, its internal latent representations are never accessed, and g_θ is evaluated only once per action chunk. The training objective aligns the corrected valid suffix with expert actions at the actual takeover time while preserving the behavior of the base policy when $d = 0$.

Specifically, an optical-flow encoder E_F and a state encoder E_s process the two inputs independently, and a fusion network G_θ predicts the residual from their concatenated features:

$$z_F = E_F(F_{t,d}), \qquad z_s = E_s(\Delta s_{t,d}), \qquad r_{t,d} = G_\theta([z_F; z_s]). \tag{4}$$

4.2 Chunk Correction

Let $j = 0, \ldots, H - d - 1$ index the valid suffix. For a delta-action policy parameterized by single-step increments, RTA adds the same residual to every action in the suffix:

$$\widetilde{a}_{d+j|t} = a_{d+j|t} + r_{t,d}. \tag{5}$$

Because incremental actions naturally accumulate during sequential execution, no additional horizon-dependent scaling is required.

For an absolute/state-action policy, the internal state difference is first projected into the action coordinate system using a mapping P provided by the robot interface:

$$\Delta s^{(a)}_{t,d} = P(s_{t+d} - s_t) \in \mathbb{R}^{D_a}. \tag{6}$$

The absolute action targets are then corrected as

$$\widetilde{a}_{d+j|t} = a_{d+j|t} + \Delta s^{(a)}_{t,d} + (j+1) r_{t,d}. \tag{7}$$

Accordingly, the first action in the suffix receives one residual correction, while subsequent absolute targets receive progressively larger offsets along the horizon. If the state and action spaces share the same coordinates and dimensionality, P is the identity map.

Algorithm 1: **Asynchronous Execution Procedure for RTA**

Input: Frozen policy π, corrector g_θ, and current action buffer $\mathcal{B}$
Output: A continuously updated safe action stream

while *the task is not terminated* **do**
  Execute the next action in $\mathcal{B}$ at the control rate;
  if *a new chunk is required and inference is idle* **then**
    Cache (I_t, s_t) and launch $A_t \sim \pi(\cdot \mid o_t, s_t, l)$ asynchronously;
  if A_t *returns at* $t + d$ **then**
    Acquire (I_{t+d}, s_{t+d}) and compute d;
    if $d < H$ *and* $\mathcal{B}$ *is safe* **then**
      $F_{t,d} \leftarrow \mathrm{Flow}(I_t, I_{t+d})$; $\Delta s_{t,d} \leftarrow s_{t+d} - s_t$;
      $r_{t,d} \leftarrow g_\theta(F_{t,d}, \Delta s_{t,d})$;
      $\mathcal{B} \leftarrow \mathcal{C}(A_t[d:], r_{t,d}, \Delta s_{t,d})$; ▷ `atomic update`
    else
      Invoke the platform-specific safe fallback;

4.3 Implementation Details

Model architecture. We resize the two global-view images to 64×64 and use the Farnebäck algorithm (Farnebäck, 2003) to compute two-channel optical flow $F_{t,d} \in \mathbb{R}^{2\times64\times64}$. Since only the approximate direction and magnitude of motion over the inference interval are required, the flow resolution can be substantially lower than the visual-input resolution used by the base VLA. The state branch takes the normalized state difference as input. The optical-flow encoder comprises three stride-2, 3×3 convolutional blocks with channel widths $2 \to 32 \to 64 \to 128$. Each block applies GroupNorm and GELU, and an AdaptiveAvgPool layer produces a 128-dimensional feature vector. The state encoder is a two-layer $8 \to 128 \to 128$ MLP with GELU activations. The resulting 128-dimensional features are concatenated and passed to a $256 \to 512 \to 512 \to 7$ residual head with GELU and Dropout in its hidden layers. RTA requires only the image, state, and action interfaces exposed by the base system; it does not access the visual tokens, language tokens, or action latents of the base policy.

Training and sampling. During training, we freeze π and optimize only θ. We sample an integer delay from the supported range $d \in \{0, \ldots, d_{\max}\}$ according to a predefined distribution. For each demonstration sample (o_t, s_t), we run the base policy offline to obtain A_t and construct the correction inputs from $(I_t, I_{t+d}, s_t, s_{t+d})$. Let $\mathcal{C}$ denote the representation-specific correction operator defined by Equation equation 5 or Equation equation 7. The loss over the corrected valid suffix is

$$\mathcal{L}_{\mathrm{RTA}} = \frac{1}{H-d} \sum_{j=0}^{H-d-1} \left\| \mathcal{C}(a_{d+j|t}, r_{t,d}, \Delta s_{t,d}) - a^*_{t+d+j} \right\|_2^2 . \tag{8}$$

Here, a^*_{t+d+j} denotes the expert action aligned with the corresponding execution time. At deployment, the base policy generates A_t using its original sampling procedure, while the correction head deterministically predicts $r_{t,d}$ in a single forward pass.

Deployment framework. Algorithm 1 summarizes the asynchronous execution procedure. The delay d is determined from the actual inference completion time, allowing RTA to accommodate latency variations without assuming a fixed delay. If $d \geq H$, the previous action chunk is nearly exhausted, or a candidate action violates a safety constraint, the controller falls back to a platform-specific safe behavior, such as holding position, slowing down, or stopping.

5 Experiments

We evaluate RTA along four dimensions. First, we assess the robustness of asynchronous execution methods to increasing inference delays on the static LIBERO benchmark. Second, we use the dynamic DOMINO benchmark to examine whether modeling relative changes mitigates prediction–execution misalignment in the presence of external motion. Third, we compare the correction overhead of our one-shot formulation with that of A2C2, a state-of-the-art per-step correction method.

Finally, we evaluate RTA on real-robot tasks that involve realistic perception, control latency, and object motion.

5.1 Static Benchmark: LIBERO

Benchmark. LIBERO (Liu et al., 2023) is a standard benchmark for multitask and lifelong robot manipulation. We evaluate on four suites of 10 tasks each. LIBERO-Spatial varies the spatial relationships among target objects while keeping the objects and manipulation goals fixed, thereby testing spatial-relation understanding. LIBERO-Object varies the target objects across similar scenes and manipulation patterns to assess object knowledge. LIBERO-Goal varies the task objectives while retaining the same objects and layouts to assess diverse manipulation behaviors. LIBERO-10, also referred to as LIBERO-Long, comprises long-horizon tasks with greater diversity in objects, layouts, and goals. We adopt SmolVLA (Shukor et al., 2025) as the VLA backbone, with a prediction horizon of $H = 50$ and an execution horizon of $K = 30$. We evaluate discrete inference delays $d \in \{0, 1, 2, 4, 8, 15, 20\}$. Further per-task results are provided in Appendix C.1.

Baselines. Sync waits for the next inference call after executing each action chunk and thus does not hide model latency; we use it as an idealized zero-inference-delay reference rather than include it in the ranking of asynchronous methods. We compare against five asynchronous baselines: (1) Naive Async directly applies the pretrained policy and always executes the next action from the most recently returned chunk; (2) RTC formulates action-chunk generation as inference-time flow inpainting, constraining the overlap between consecutive chunks with the actions committed for execution during inference (Black et al., 2026); (3) Training-time RTC learns to generate the remaining suffix conditioned on a known action prefix, avoiding additional backpropagation at deployment (Black et al., 2025); (4) VLASH rolls the robot state forward to the estimated handover time and conditions the new action chunk on this predicted execution state (Tang et al., 2025); and (5) A2C2 predicts an action residual at every control step from the latest observation, the current base action, and the position within the chunk (Sendai et al., 2025). By contrast, RTA performs a single correction to the valid suffix of each new action chunk before handover, using the relative changes observed during the inference window. During training, we use a maximum delay of $d_{\max} = 8$.

Results. Figure 3 reports success rate as a function of inference delay. All asynchronous methods degrade to varying degrees as d increases. With longer inference delays, each new action chunk is conditioned on an increasingly stale observation, while the robot advances farther along the preceding chunk before handover. The resulting mismatch between the prediction and execution intervals becomes increasingly severe. This degradation is especially pronounced for Naive Async, RTC, and Training-time RTC. RTA, by contrast, achieves the highest average success rate across the four suites and evaluated delays, with a more gradual decline at large delays. These results indicate that one-shot correction based on relative changes provides strong robustness to inference latency. Notably, although RTA is trained only with delays of up to eight steps ($d_{\max} = 8$), it retains strong performance at the unseen delays $d = 15$ and $d = 20$, demonstrating that the learned correction rule extrapolates beyond the training delay range.

5.2 Dynamic Benchmark: DOMINO

Benchmark. DOMINO (Fang et al., 2026) evaluates dynamic robot manipulation in the presence of moving objects. We use the Level 1 setting, in which target objects move at constant velocities sampled from a predefined range. This setting tests whether a policy can estimate an object's instantaneous state and adapt to its linear motion. Our VLA backbone is $\pi_{0.5}$ (Intelligence et al., 2025) trained on the DOMINO dataset. We fix the prediction horizon to $H = 50$ and the execution horizon to $K = 30$ for all methods. Sync is evaluated at $d = 0$, whereas all asynchronous methods are evaluated at $d = 8$ and must account for both external object motion and prediction–execution misalignment. We report Success Rate (SR) and Route Completion (RC). SR denotes the percentage of episodes that satisfy all task conditions within the time budget. RC measures spatial convergence through the change in distance between the end effector and the target object, with successful episodes assigned an RC of 100.

Results. Under the same asynchronous setting ($H = 50$, $K = 30$, and $d = 8$), RTA improves SR and RC over Naive Async by 3.55 and 3.41 percentage points, respectively. Despite operating with a delay of $d = 8$, RTA matches the idealized $d = 0$ Sync reference in both metrics. These results suggest that, even when the base VLA observation is stale by the time its action chunk becomes

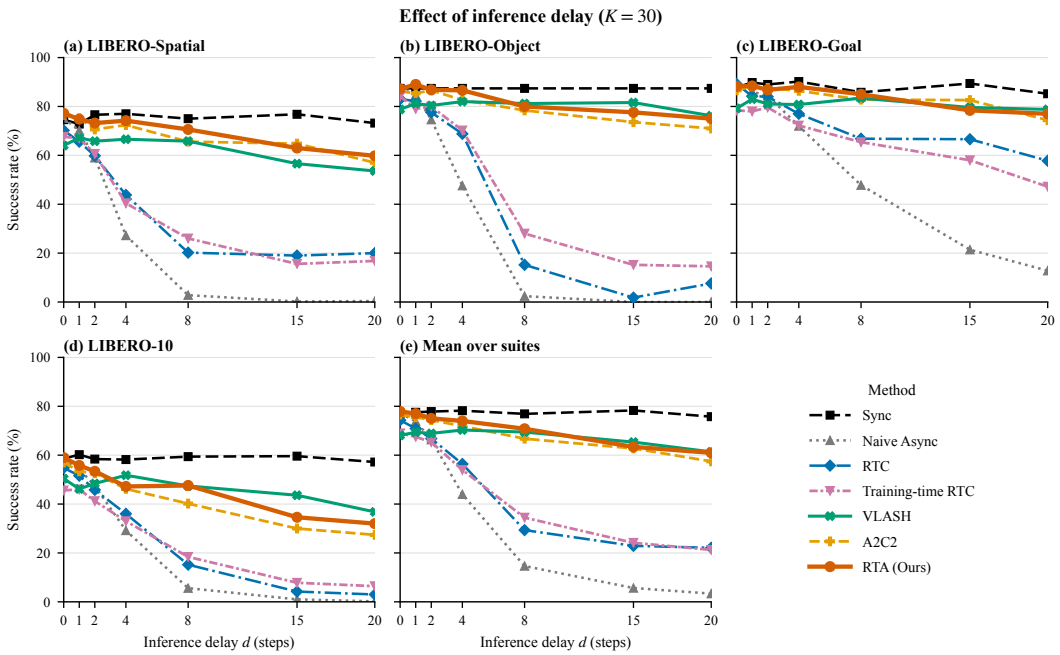


Figure 3: Success rates on the four LIBERO suites under varying inference delays ($H = 50$, $K = 30$). RTA exhibits more gradual performance degradation as the delay increases.

Table 1: Dynamic manipulation performance of $\pi_{0.5}$ under different execution schemes on DOMINO Level 1, with prediction horizon $H = 50$ and execution horizon $K = 30$. Sync is evaluated at $d = 0$, whereas asynchronous methods are evaluated at $d = 8$. We report success rate (SR, %) and route completion (RC, %); higher is better, and the best results are bolded.

Method	d	SR (%) ↑	RC (%) ↑
Sync	0	11.71	33.23
Naive Async	8	8.74	29.94
A2C2	8	3.40	19.97
RTA (Ours)	8	**12.29**	**33.35**

available, RTA can infer the relative displacement between the robot and the moving target from visual and robot-state changes over the inference window. Correcting the pending actions with this displacement effectively mitigates prediction–execution misalignment in dynamic environments.

5.3 Correction Cost: A2C2 vs. RTA

Table 2 compares the computational overhead of the A2C2 and RTA correction modules. We conduct all measurements on an NVIDIA GeForce RTX 5090 with a batch size of 1 and $K = 30$, using 20 warm-up iterations followed by 100 timed trials. A2C2 performs a separate correction at each of the 30 control steps in a chunk, whereas RTA performs a single correction before handover. We therefore aggregate correction overhead over a complete action chunk to ensure a consistent comparison between the two paradigms.

Results. RTA's one-shot chunk correction substantially reduces the cumulative overhead of per-step correction. In FP32, RTA requires approximately $1/63$ of A2C2's model memory, $1/20{,}315$ of its per-chunk FLOPs, and $1/368$ of its P95 correction latency. The corresponding FP16 ratios are approximately $1/63$, $1/19{,}032$, and $1/434$. Across both precisions, RTA therefore reduces per-chunk computation by roughly four orders of magnitude and P95 latency by roughly two orders of magnitude, highlighting the efficiency of correcting only once at handover.

5.4 Real-Robot Experiments

Setup and Task Design. We conduct real-robot experiments on the Agilex Cobot Magic platform. The VLA runs on a server equipped with an NVIDIA GeForce RTX 5090 GPU, while the correction module runs on a local computer equipped with an NVIDIA GeForce RTX 4060 GPU. We consider

Table 2: Per-chunk correction overhead of A2C2 and RTA, measured on an NVIDIA GeForce RTX 5090 with batch size 1 and $K = 30$. We report the number of corrections, model memory, cumulative neural-network FLOPs, and P95 correction latency per chunk after 20 warm-up iterations and over 100 timed trials.

Method	Precision	Corrections/chunk	Model memory (MiB)	Neural FLOPs/chunk	P95 latency (ms)
A2C2	FP32	30	121.98	424.17 G	426.34
A2C2	FP16	30	60.99	779.16 G	550.97
RTA (Ours)	FP32	1	1.94	20.88 M	1.16
RTA (Ours)	FP16	1	0.97	40.94 M	1.27

Table 3: Success rates of Sync (OpenPI), Naive Async, and RTA on two real-robot dynamic grasping tasks: grasping a conveyor-belt block moving at a constant speed of $2.5\,\mathrm{cm/s}$ and grasping a can rolling at a nonuniform speed. The final column reports the average success rate across the two tasks. Higher is better, and the best results are bolded.

Method	Conveyor-block SR (%)	Rolling-can SR (%)	Mean SR (%)
Sync (OpenPI)	2.0	6.0	4.0
Naive Async	34.0	20.0	27.0
RTA (Ours)	**54.0**	**28.0**	**41.0**

two dynamic grasping tasks: (1) grasping a block moving at a constant speed of 2.5 cm/s on a conveyor belt, which evaluates temporal alignment under regular translational motion; and (2) grasping a can rolling at a nonuniform speed, which evaluates adaptation to more complex variations in target speed and direction. We collect 100 demonstrations per task and separately fine-tune all parameters of $\pi_{0.5}$ (Intelligence et al., 2025) on each dataset. We compare RTA against two baselines: the synchronous execution implementation in the official OpenPI repository (Physical Intelligence, 2025), denoted as Sync (OpenPI), and naive asynchronous execution without action correction, denoted as Naive Async. We use task success rate as the evaluation metric.

Results. RTA achieves the highest success rate on both tasks. For the block moving at a constant speed, relative changes over the inference window provide stable cues about the target's motion direction and displacement, enabling RTA to compensate for its updated position before the new action chunk takes control. For the nonuniformly rolling can, the visual and state differences recomputed at the end of inference capture variations in speed and direction that are unavailable in the initial observation. RTA therefore adapts more effectively than execution schemes that rely solely on stale observations. Overall, these results demonstrate that RTA improves temporal alignment under both regular and nonuniform object motion.

The low success rate of Sync (OpenPI) should not be interpreted as contradicting the idealized synchronous reference used in simulation. In simulation, Sync is evaluated at $d = 0$ to isolate algorithmic effects, whereas the real-system implementation blocks action execution while network communication and model inference proceed. Because the target continues moving during this waiting period, the observation used to generate the next chunk is already stale when execution resumes, even though no asynchronous chunk handover occurs. Thus, blocking synchronization removes the old-to-new chunk handover mismatch but does not eliminate observation–execution latency in a dynamic environment. Naive Async avoids this idle period but still executes actions conditioned on an earlier observation, whereas RTA maintains continuous control and corrects the returned chunk using the relative robot–environment change accumulated during inference. Representative execution sequences illustrating these differences are provided in Appendix Figure B1.

6 Conclusion

We identify stale-observation-induced prediction–execution misalignment as a key limitation of asynchronously executed, action-chunking VLA policies. To address it, we introduce RTA, a lightweight correction module that leverages relative robot–environment changes while keeping the base policy frozen. Across static and dynamic benchmarks in both simulation and real-world settings, RTA improves robustness to variable inference delays, generalizes beyond training delays, and better adapts to environmental motion, with only 20.88 MFLOPs and 1.16 ms P95 correction latency.

AI USE STATEMENT

Large language models were used to assist with language polishing and stylistic refinement, primarily to improve clarity, grammatical accuracy, coherence, and adherence to academic writing conventions. Large language models were not used to formulate the research content, design the experiments, or analyze the results; all technical contributions, mathematical definitions, and experimental results were developed and produced by the authors. All model-generated content was reviewed, revised, and approved by the authors.

ETHICS STATEMENT

This study does not involve human subjects or personally sensitive data. All real-robot experiments are conducted on an Agilex Cobot Magic platform in a controlled laboratory, using a predefined workspace, platform-level safety constraints, emergency-stop mechanisms, and human supervision. RTA mitigates execution errors caused by stale observations but is not a safety controller and cannot eliminate perception, model, or physical-motion risks.

REPRODUCIBILITY STATEMENT

We support reproducibility through detailed method and experiment specifications. Section 4 and Algorithm 1 describe the delay model, relative-change representation, action correction, training objective, and asynchronous execution procedure; Section 4.3 specifies the network architecture and input processing; and Section 5 reports benchmark delays, baselines, metrics, and timing protocols.

REFERENCES

Ayoub Agouzoul. Understanding asynchronous inference methods for vision-language-action models. *arXiv preprint arXiv:2605.08168*, 2026.

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.

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

Kevin Black, Manuel Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. *Advances in Neural Information Processing Systems*, 38:33383–33407, 2026.

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, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 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*, 44(10-11):1684–1704, 2025.

Heng Fang, Shangru Li, Shuhan Wang, Xuanyang Xi, Dingkang Liang, and Xiang Bai. Towards generalizable robotic manipulation in dynamic environments. In *European Conference on Computer Vision*, pp. 314–333. Springer, 2026.

Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on Image analysis*, pp. 363–370. Springer, 2003.

Denis Fortun, Patrick Bouthemy, and Charles Kervrann. Optical flow modeling and computation: A survey. *Computer Vision and Image Understanding*, 134:1–21, 2015.

Yutong Hu, Jan-Nico Zaech, Nikolay Nikolov, Yuanqi Yao, Sombit Dey, Giuliano Albanese, Renaud Detry, Luc Van Gool, and Danda Paudel. Ar-vla: True autoregressive action expert for vision-language-action models. *arXiv preprint arXiv:2603.10126*, 2026.

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.

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.

Sunwoo Kim, Jeongjun Kim, and Joseph Lim. Time optimal execution of action chunk policies beyond demonstration speed. In *International Conference on Learning Representations*, volume 2026, pp. 154823–154842, 2026.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

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.

Yueen Ma, Zixing Song, Yuzheng Zhuang, Jianye Hao, and Irwin King. A survey on vision–language–action models for embodied ai. *IEEE Transactions on Neural Networks and Learning Systems*, 2026.

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.

Physical Intelligence. openpi: Open-source models and packages for robotics. https://github.com/Physical-Intelligence/openpi, 2025. GitHub repository.

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.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.

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

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *European conference on computer vision*, pp. 402–419. Springer, 2020.

Quan Vuong, Sergey Levine, Homer Rich Walke, Karl Pertsch, Anikait Singh, Ria Doshi, Charles Xu, Jianlan Luo, Liam Tan, Dhruv Shah, et al. Open x-embodiment: Robotic learning datasets and rt-x models. In *Towards Generalist Robots: Learning Paradigms for Scalable Skill Acquisition@ CoRL2023*, 2023.

Haoxuan Wang, Gengyu Zhang, Yan Yan, Yuzhang Shang, Ramana Kompella, and Gaowen Liu. Real-time robot execution with masked action chunking. In *International Conference on Learning Representations*, volume 2026, pp. 120161–120179, 2026.

Haozhe Xie, Beichen Wen, Jiarui Zheng, Zhaoxi Chen, Fangzhou Hong, Haiwen Diao, and Ziwei Liu. Dynamicvla: A vision-language-action model for dynamic object manipulation. *arXiv preprint arXiv:2601.22153*, 2026.

Han Xue, Jieji Ren, Wendi Chen, Gu Zhang, Yuan Fang, Guoying Gu, Huazhe Xu, and Cewu Lu. Reactive diffusion policy: Slow-fast visual-tactile policy learning for contact-rich manipulation. *arXiv preprint arXiv:2503.02881*, 2025.

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.

Yixiang Zhu, Yonghao Chen, Zijie Yang, Yusong Hu, and Xinyu Chen. Deflect: Temporal counterfactual preference learning for delay-robust asynchronous vlas, 2026. URL https://arxiv.org/abs/2605.19294.

A Hardware Setup

We conduct the real-robot experiments on the dual-arm Agilex Cobot Magic platform. As shown in Figure A1, we use only one arm for manipulation and reserve the region on the non-executing-arm side as the entry path for target objects. In the conveyor-belt block task, the block enters the executing arm's workspace from this side at a constant speed; the rolling can enters the manipulation area from the same direction. This layout provides an observable target trajectory before the object

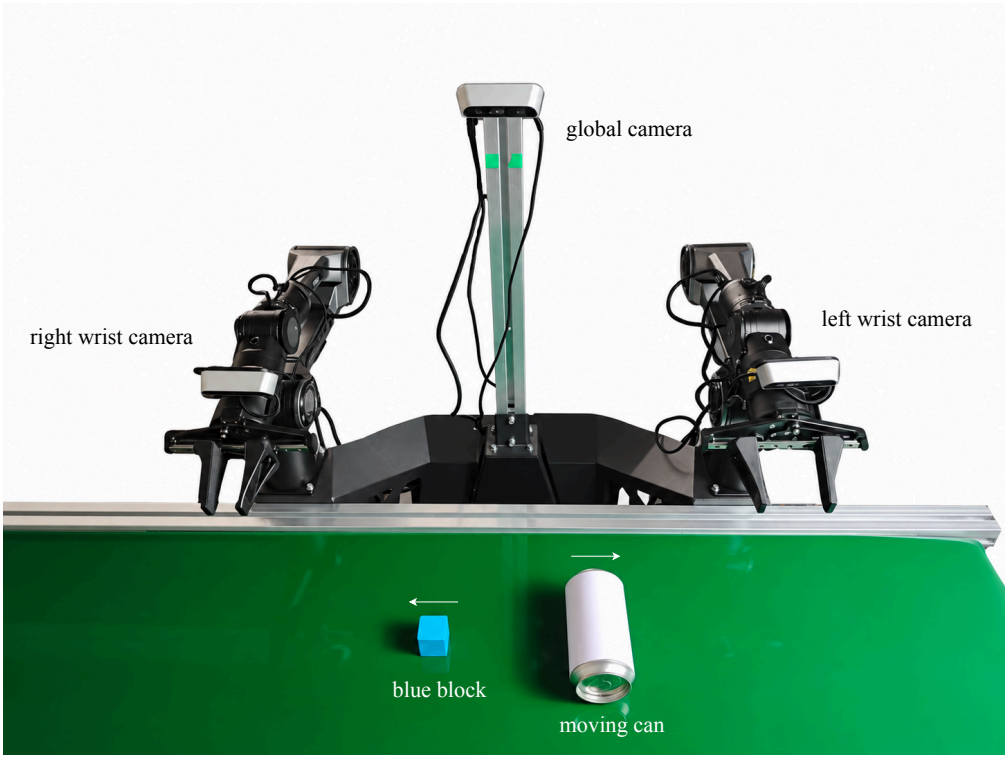


Figure A1: Real-robot experimental platform and system architecture. Dynamic grasping is performed with one arm of the Agilex Cobot Magic platform, while target objects enter the manipulation area from the side of the non-executing arm. The base VLA runs on a server equipped with an NVIDIA GeForce RTX 5090 GPU, and the RTA correction module runs on a local computer equipped with an NVIDIA GeForce RTX 4060 GPU.

reaches the primary grasping region, giving the global camera time to capture the target's motion and the control system time to perform recognition, inference, and action correction. It also prevents the unused arm from introducing additional occlusions into the camera view or manipulation workspace.

The system follows a client–server architecture. The base VLA policy $\pi_{0.5}$ (Intelligence et al., 2025) runs on a server equipped with an NVIDIA GeForce RTX 5090 GPU and generates action chunks conditioned on observations, robot states, and language instructions. The RTA correction module runs on a local computer equipped with an NVIDIA GeForce RTX 4060 GPU and computes action residuals from the global images and robot states captured at the two endpoints of the inference window before each new action chunk takes control. This architecture decouples the computationally intensive base-policy inference from local control while preserving the realistic, non-ideal latencies introduced by network communication, image processing, and system scheduling.

B TASK EXAMPLES

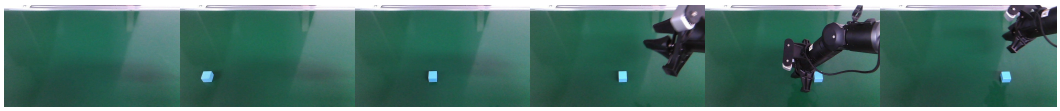
Figure B1 presents four representative sequences of global-camera observations from the conveyor-belt block task, illustrating the effect of stale observations on grasping behavior and the correction achieved by RTA. Each sequence shows the temporal evolution of the relative position between the block and the gripper.

Figure B1(a) shows a representative failure of the official synchronous deployment code from OpenPI (Physical Intelligence, 2025). Here, Sync denotes blocking synchronous execution in the real system, including the time spent on network communication and model inference; it therefore differs from the idealized $d = 0$ Sync reference used in simulation to isolate algorithmic factors. While the robot waits for a new action chunk, the block continues to move along the conveyor belt. When the arm subsequently attempts the grasp, the gripper closes at a location that no longer aligns with the block's current position, resulting in failure. Thus, although blocking synchronous execution avoids the direct old-to-new chunk handover characteristic of asynchronous execution, real inference latency still creates a temporal mismatch between observation and execution in dynamic environments.

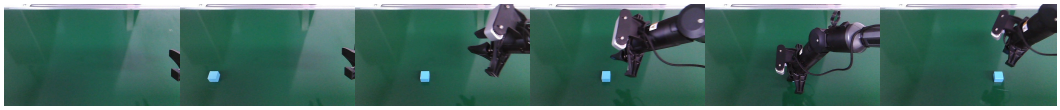
Figure B1(b) shows a failure of Naive Async without a correction module. This approach continues to execute buffered actions while the base VLA performs inference, but the new action chunk remains conditioned on the observation captured at the beginning of inference. The sequence shows that the block continues to move along the conveyor belt during inference and action handover, whereas the gripper attempts to grasp at a position that does not align with the block's current location. The grasp consequently fails. This example illustrates that continuous execution alone cannot eliminate the prediction–execution misalignment caused by stale observations.

Figure B1(c) shows a successful execution by RTA. RTA compares the global images and robot states at the beginning and end of inference and predicts action residuals from the relative changes observed over the inference window. In this sequence, the gripper closes near the block's updated position and successfully completes the grasp. This outcome is consistent with RTA's objective of correcting pending actions using the relative changes that occur during inference.

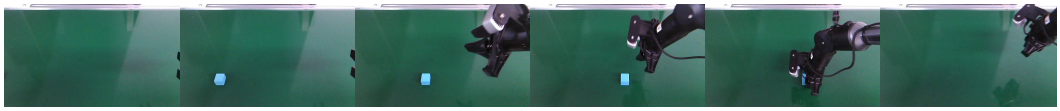
Figure B1(d) presents a failure case of RTA. In this sequence, the gripper remains insufficiently aligned with the block when it closes. The final frame shows a marked change in the block's orientation, indicating that the gripper made contact but failed to secure the object. One possible explanation is the nonzero latency of the RTA correction head, during which the block continues to move. Because the current method does not explicitly compensate for the additional displacement accumulated during this correction stage, the resulting adjustment may be insufficient. This case highlights a limitation of one-shot residual correction: when the target undergoes substantial displacement over the inference window, motion estimation is inaccurate, or correction latency is non-negligible, the corrected grasp position may still fail to align adequately with the target.



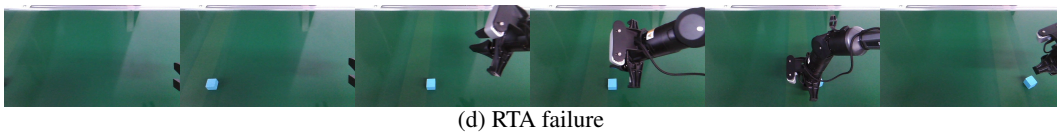
(a) Sync (OpenPI) failure



(b) Naive Async failure



(c) RTA success



(d) RTA failure

Figure B1: Representative execution sequences from the conveyor-belt block task. (a) Sync (OpenPI) failure: inference-induced waiting in the real system allows the block to move after observation, causing the gripper to close at a misaligned position. (b) Naive Async failure: continuous execution does not eliminate the grasping error caused by stale observations. (c) RTA success: the gripper closes near the block's updated position and successfully grasps it. (d) RTA failure: the gripper contacts and rotates the block but fails to secure it.

C PER-TASK SIMULATION RESULTS

C.1 LIBERO

Figure C1 presents the complete per-task results across the four LIBERO suites. The first group of panels corresponds to LIBERO-Spatial and LIBERO-Object, while the second corresponds to LIBERO-Goal and LIBERO-10. All methods use the same base policy and evaluation protocol, with a prediction horizon of $H = 50$, an execution horizon of $K = 30$, and discrete inference delays $d \in \{0, 1, 2, 4, 8, 15, 20\}$. RTA is trained with a maximum delay of $d_{\max} = 8$.

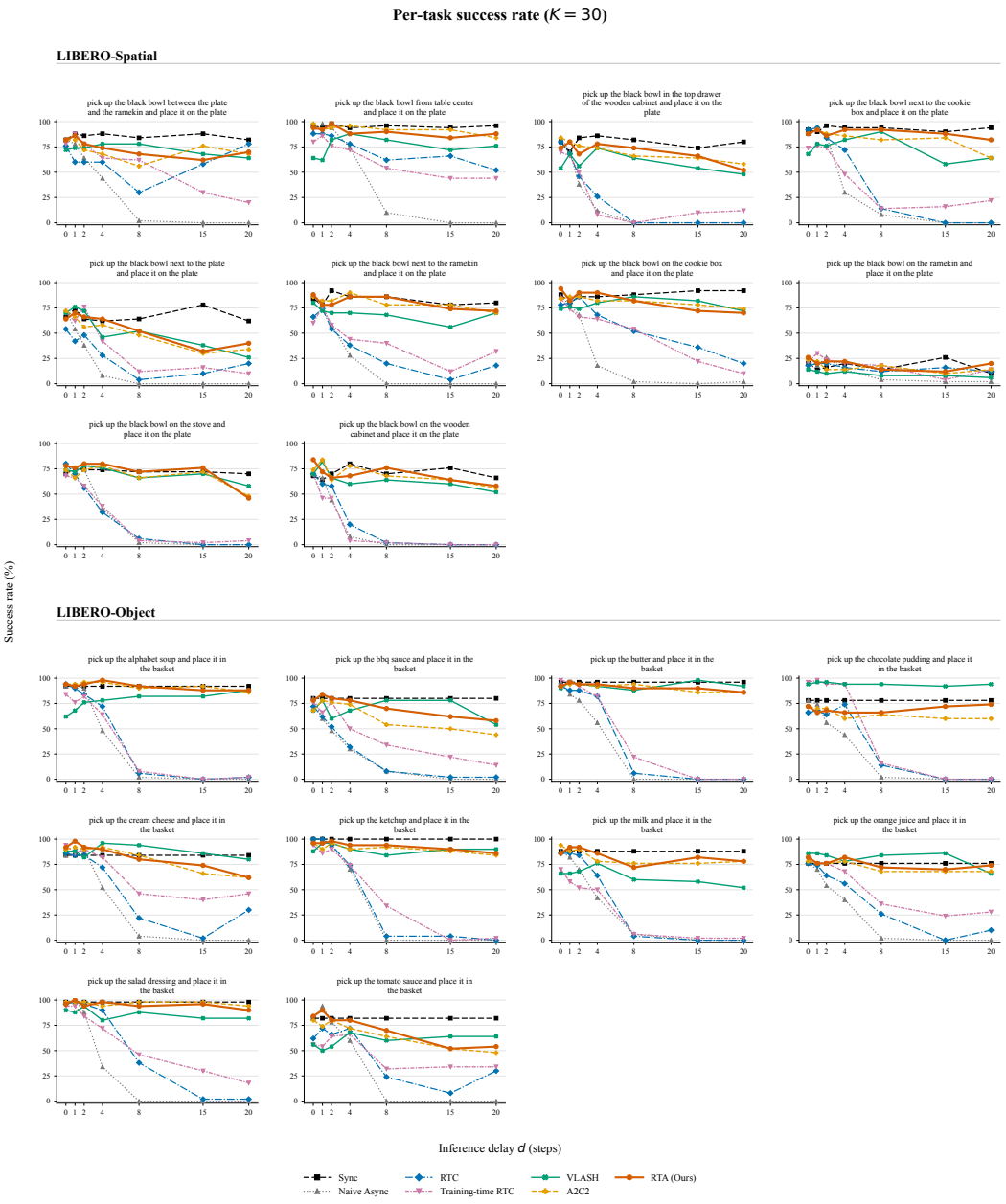


Figure C1: Per-task success rates on LIBERO under discrete inference delays $d \in \{0, 1, 2, 4, 8, 15, 20\}$, with a fixed prediction horizon of $H = 50$ and execution horizon of $K = 30$. This group of panels reports per-task results for LIBERO-Spatial and LIBERO-Object. RTA is trained with a maximum delay of $d_{\max} = 8$.

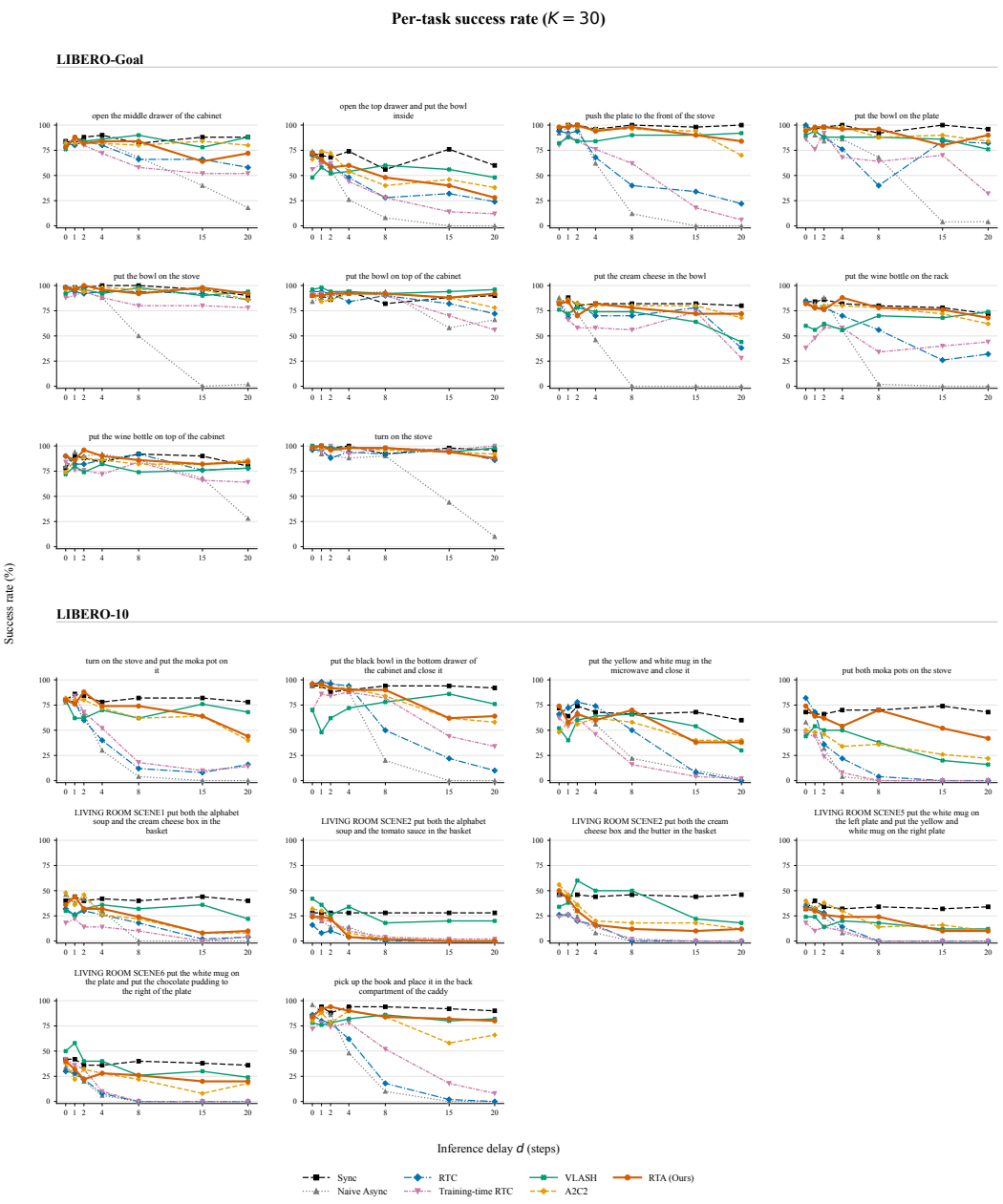


Figure C1: Per-task success rates on LIBERO under the same evaluation setting (continued). This group of panels reports per-task results for LIBERO-Goal and LIBERO-10, completing all 40 tasks across the four suites.

Table C1: Per-task success rate (SR, %) and route completion (RC, %) of $\pi_{0.5}$ under four execution schemes on 35 unseen tasks from DOMINO Level 1, with 100 episodes evaluated per task. All methods use a prediction horizon of $H = 50$ and an execution horizon of $K = 30$; Sync uses $d = 0$, whereas all asynchronous methods use $d = 8$. The best SR and RC for each task are shown in bold.

	Sync		Naive Async		A2C2		RTA	
Task	SR	RC	SR	RC	SR	RC	SR	RC
adjust bottle	**45**	**64**	**45**	59	8	29	38	55
beat block hammer	6	**35**	4	32	4	25	**7**	**35**
click alarmclock	6	14	4	13	7	14	**8**	**16**
click bell	1	**17**	**2**	**17**	1	14	1	**17**
dump bin bigbin	9	22	8	18	0	9	**15**	**29**
grab roller	**44**	**63**	40	59	29	47	**44**	62
handover block	0	48	**2**	50	0	35	**2**	**53**
handover mic	**21**	**43**	14	39	4	26	18	**43**
hanging mug	**7**	29	4	**32**	0	18	**7**	25
move can pot	**13**	**42**	6	37	2	23	12	**42**
move pillbottle pad	**1**	**18**	0	**18**	0	10	0	17
move playingcard away	**9**	**23**	6	22	1	15	6	**23**
move stapler pad	0	**22**	**2**	21	0	11	1	21
place a2b left	4	**30**	2	25	5	23	**7**	**30**
place a2b right	2	**31**	**4**	29	2	19	**4**	**31**
place bread basket	5	26	3	23	3	13	**10**	**29**
place bread skillet	3	27	6	27	0	27	**11**	**34**
place can basket	5	**37**	5	35	1	22	**7**	36
place container plate	37	48	11	23	6	14	**40**	**49**
place empty cup	**4**	**21**	0	17	1	11	0	16
place fan	1	12	1	**13**	0	8	**2**	12
place mouse pad	0	**22**	0	19	0	12	0	21
place object basket	15	**42**	10	32	2	23	**16**	39
place object scale	**1**	**23**	**1**	22	0	12	**1**	22
place object stand	0	17	0	15	2	11	**5**	**21**
place phone stand	**2**	30	0	**31**	0	20	1	28
place shoe	23	43	10	29	6	19	**31**	**50**
press stapler	**9**	**25**	3	19	6	19	7	23
put bottles dustbin	20	38	**22**	**44**	5	27	19	41
put object cabinet	**17**	**54**	8	48	0	29	12	51
rotate qrcode	6	28	1	20	0	22	**9**	**30**
scan object	1	27	1	29	**2**	**35**	**2**	28
shake bottle	**44**	**50**	33	42	11	20	39	47
shake bottle horiz.	40	59	**47**	**62**	11	22	46	61
stamp seal	**9**	**34**	1	29	0	15	2	29
Avg.	11.7	33.2	8.7	29.9	3.4	19.9	**12.3**	**33.4**

C.2 DOMINO LEVEL 1

Table C1 reports per-task results for four execution schemes on 35 unseen tasks from DOMINO Level 1, with 100 episodes evaluated per task. All methods use a prediction horizon of $H = 50$ and an execution horizon of $K = 30$. Sync is evaluated at $d = 0$, whereas Naive Async, A2C2, and RTA are evaluated at $d = 8$.

D Effect of the Execution Horizon on LIBERO Performance

Figure D1 examines how the execution horizon affects asynchronous execution performance. With the prediction horizon fixed at $H = 50$ and the discrete inference delay fixed at $d = 8$, we compare the success rates of different methods on LIBERO for $K \in \{10, 20, 30, 50\}$. The figure first presents per-task results for the four suites, followed by the mean success rate within each suite and the overall mean across all four suites. A smaller K increases the action-chunk update frequency, allowing fresh observations to influence control more quickly. In contrast, a larger K reduces action-chunk transitions and preserves action continuity over a longer interval, but also extends the period during which the policy executes open-loop actions from the same chunk.

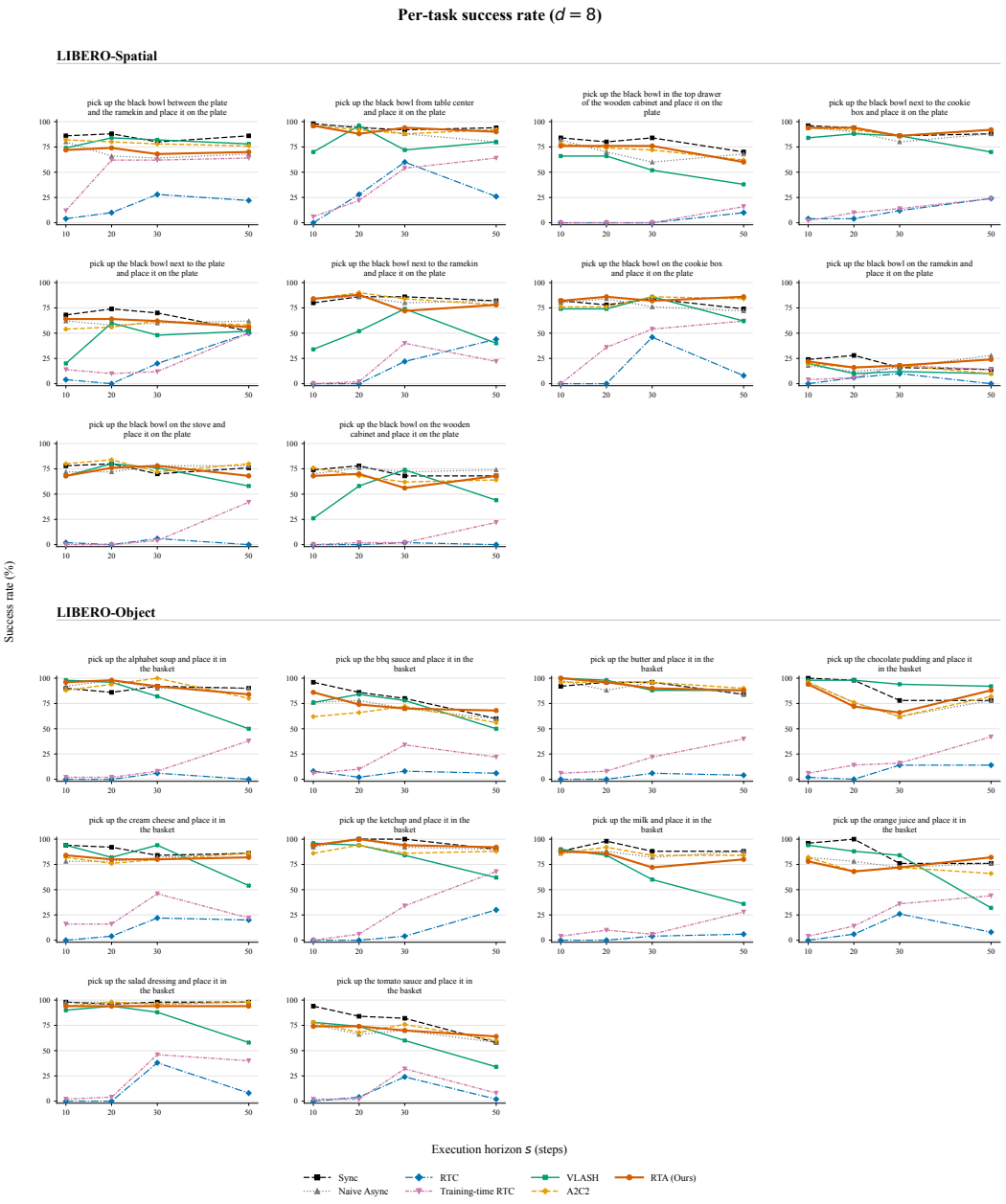


Figure D1: LIBERO success rates as a function of the execution horizon $K \in \{10, 20, 30, 50\}$, with a fixed prediction horizon of $H = 50$ and discrete inference delay of $d = 8$. This group of panels reports per-task results for LIBERO-Spatial and LIBERO-Object.

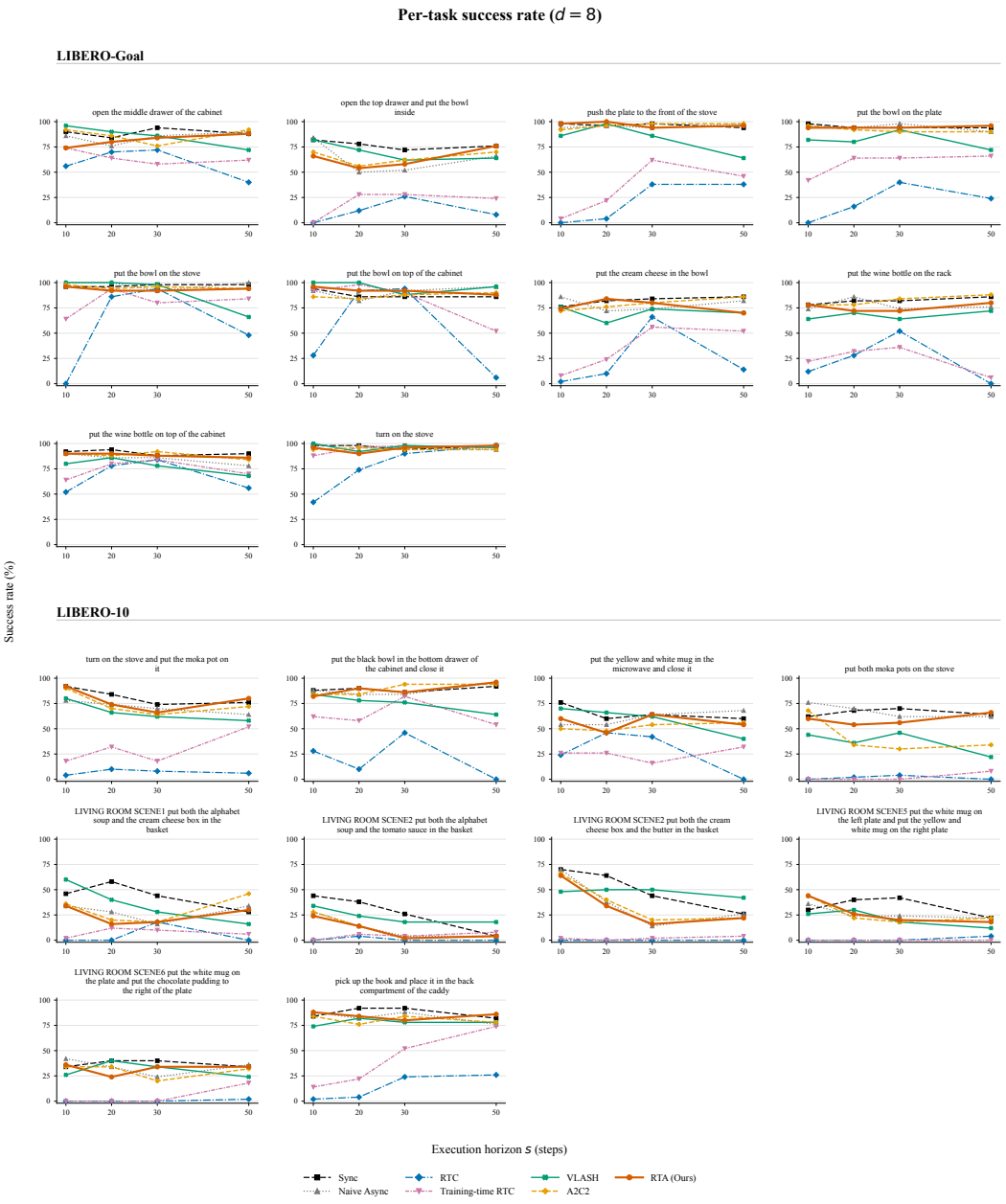


Figure D1: LIBERO success rates under the same execution-horizon setting (continued). This group of panels reports per-task results for LIBERO-Goal and LIBERO-10, completing all 40 tasks across the four suites.

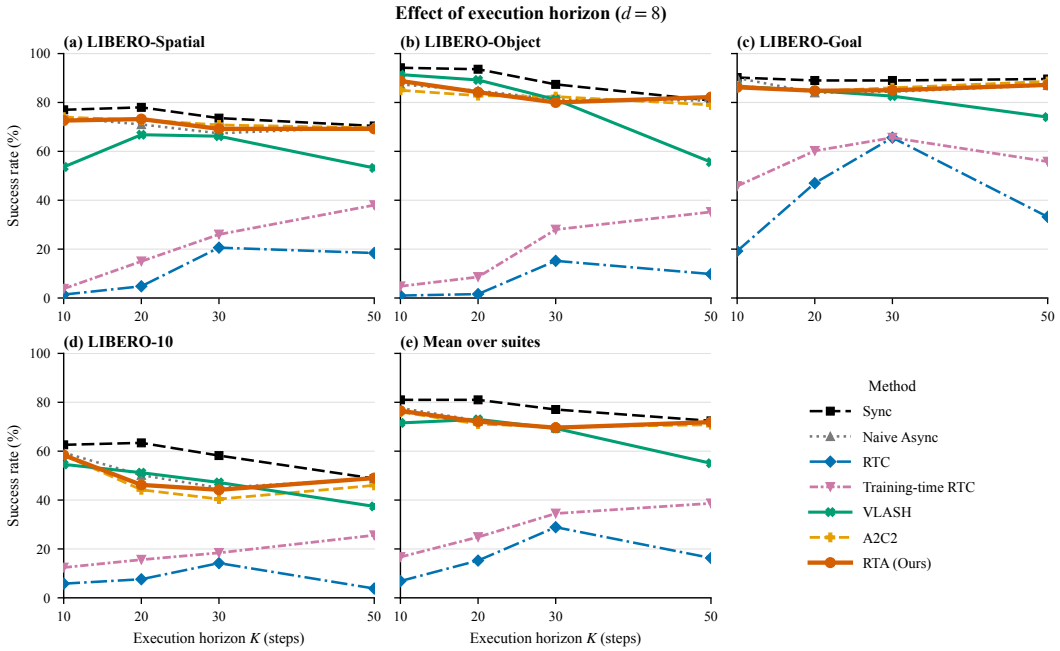


Figure D1: LIBERO success rates under the same execution-horizon setting (continued). Panels (a)–(d) report the mean success rate within each of the four suites, while panel (e) reports the mean across all four suites.

E LIMITATIONS

Our method is not without limitations. First, the visual branch of RTA relies on optical flow to represent scene changes over the inference window, but optical-flow estimation is sensitive to illumination changes, reflections, occlusions, motion blur, and low-texture regions, potentially yielding action residuals with inaccurate directions or magnitudes (Fortun et al., 2015). Second, optical-flow computation, state-difference processing, and residual-network inference introduce additional latency, while the current method does not explicitly compensate for environmental changes that continue to accumulate during this interval; consequently, it may underestimate the required correction when the target moves rapidly. Future work could investigate more robust motion representations, such as RAFT (Teed & Deng, 2020), augmented with depth, object tracking, or confidence estimation, as well as latency-aware residual calibration based on measured correction latency and motion trends.