WHEN TO LAUNCH INFERENCE FOR ACTION CHUNKING UNDER LATENCY?

Anonymous authors
Paper under double-blind review

ABSTRACT

Policies parameterized by expressive generative models promise a powerful interface for real-time decision making. To boost performance and promote execution smoothness, these models predict a chunk of future actions in a single inference call. Yet, the non-negligible inference latency introduces a delicate trade-off. Early inference may prevent the model from reacting to critical information revealed shortly afterward; late inference suffers from staleness before the next action chunk prediction takes effect. Existing work mainly improves the chunk prediction policy itself, but sidesteps the equally essential question: when to launch inference under latency? This paper bridges this gap by formulating inference timing as a meta control policy in a semi-MDP induced by the original decision-making problem. We show that value can vary non-monotonically with inference timing, illustrate its consequences in a hidden-danger example, and adapt a fitted Q-iteration algorithm to learn the meta control policy. Theoretically, we prove a performance bound that disentangles sources of sub-optimality, notably including action chunk generation quality and fitted Q estimation error. Empirically, experiments on Kinetix and DOM span different environments and action-policy scales. Adding learned launch timing improves task success while reducing action-model calls compared with the same action generators under their default launch schedules, with larger gains on harder tasks and abrupt dynamic adaptation.

1 INTRODUCTION

Generative action policies based on diffusion and flow matching models have become increasingly attractive for physical control problems (Janner et al., 2022; Chi et al., 2025; Funk et al., 2024; Black et al., 2024; Physical Intelligence et al., 2025; Hou et al., 2025; Kim et al., 2026). In many real-time decision-making settings, these models are used as action-chunking policies: a single inference call returns a short sequence of future actions, which is stored in an action buffer and consumed over subsequent control steps (Zhao et al., 2023; Black et al., 2025a). By generating temporally coherent chunks, these models can represent multimodal demonstrations and produce smooth behavior over multiple future steps (Lee et al., 2024). This expressiveness, however, often comes with non-negligible inference latency, due to the iterative sampling nature of diffusion and flow models (Prasad et al., 2024; Jiang et al., 2025).

The presence of latency complicates the effectiveness of an action chunk (Black et al., 2025a; Tang et al., 2025; Wang et al., 2026b). A chunk requested at the current time is generated from the current state, but it may not affect action execution immediately. By the time the chunk returns, some near-term actions are already committed, while the finite chunk length also limits how far into the future the returned sequence can be effective (Black et al., 2025a; Liang et al., 2026). Thus the useful part of a returned chunk is determined jointly by the inference delay, chunk length, fixed buffer-update rule, and current action buffer (Black et al., 2025b; Liu et al., 2026b). Launching early allows a new chunk to affect execution sooner, but may base future actions on incomplete information. Waiting can reveal that information, but also postpones the first time at which the new chunk can affect behavior.

This creates two coupled design objects. The first is the action policy π_{chunk}, which maps the current state-buffer context to a candidate action chunk. The second is the meta control policy π_{meta}, which decides when the system calls π_{chunk}. This paper focuses on the second object. We treat the quality of π_{chunk} as a crucial input, but ask when a meta controller should launch a chunk generator.

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.

Existing work on action chunking mostly centers on π_{chunk}. Many methods improve the generator or its interface to the buffer, for example by conditioning on buffered actions, preserving or inpainting committed prefixes, adapting chunk contents or execution horizons, or training under delayed-action context (Black et al., 2025a;b; Randazzo, 2026; Wang et al., 2026b; Liang et al., 2026; Wang et al., 2026a; Nie et al., 2026). More recently, several works have begun to move toward the meta control layer, for example by adaptively downsampling generated action chunks (Wu et al., 2026), allocating reasoning and inference compute (Liu et al., 2026a; Li et al., 2026), or triggering replanning through lightweight execution verification (Wang et al., 2026c). However, the timing of action chunk generation often remains hand-coded. In particular, many systems call the generator as soon as the previous inference has returned.

The immediate launch rule is natural, but it can be costly and sub-optimal. Each query to a diffusion or flow-based action generator can require substantial computation (Prasad et al., 2024; Jiang et al., 2025), so launching immediately after every returned inference pays this cost even when a new chunk is not yet necessary. More importantly, immediate launching can also undermine control performance in highly dynamic or safety-critical settings. A moving target may abruptly change direction, or a robot may suddenly discover a hidden danger along its route. A chunk launched just before such a change cannot use the newly available information; a later launch can adapt the motion or choose a detour. Yet waiting too long delays the response. Control value can therefore vary non-monotonically with launch time, even when the action generator is fixed. This raises the central question:

How should a controller decide when to launch action generation under latency?

In this paper, we formulate launch timing as a meta control problem over a fixed action generator. The physical system is a finite-horizon MDP in which π_{chunk} predicts action chunks; the meta controller π_{meta} observes a state summary containing the physical state, action buffer, and delay status, and decides whether to launch a new inference. A launch occupies a state-dependent number of physical steps, during which no further launch decision is available and the system continues executing buffered actions. Hence meta decisions occur at irregular steps, with each decision inducing both an accumulated reward and a variable holding time. This naturally yields a decision-stage SMDP for learning π_{meta}. Our contributions are as follows.

- We reveal a non-monotonic effect of inference timing in Section 4.1: launching earlier is not always better, because premature calls can lock the controller into stale chunks before critical information arrives, while waiting too long again degrades value. We also demonstrate the benefit of launch timing in a simulated example in Section 4.2.

- We formulate inference timing around a fixed action generator as a decision-stage SMDP (Section 3). This construction enables fitted Q-iteration to learn a Wait/Launch meta policy, and we prove a performance bound separating action-generator quality, data coverage, and estimation error (Section 5).

- We demonstrate consistent success-rate gains across delay settings and action-policy backbones in Section 6. On Kinetix, learned timing reduces action-model calls by 28.08% overall while improving success, with larger gains on harder tasks. On DOM's Dynamic Adaptation task, it raises success rates by 47.37–61.00% relative to DynamicVLA under its default launch schedule while also reducing calls, showing the value of launch timing under abrupt motion changes.

2 Related Work

Diffusion policies and action chunking. Diffusion models have recently emerged as a powerful generative paradigm for sequential decision making, modeling plans or actions (Janner et al., 2022; Ajay et al., 2022; Wang et al., 2022; Chi et al., 2025). In robot control, this paradigm is often coupled with action chunking: rather than predicting a single control at each inference step, the policy predicts a short horizon of future actions, which improves temporal consistency and supports receding-horizon execution (Zhao et al., 2023; Chi et al., 2025). Action chunking has therefore become a standard interface across a range of policy classes, including transformer and variational action chunking models (Zhao et al., 2023; George & Farimani, 2023), diffusion policies (Chi et al., 2025; Zhao et al., 2024; Octo Model Team et al., 2024; Ze et al., 2024; Hou et al., 2025), flow-matching policies (Lipman et al., 2022; Black et al., 2024), discrete or tokenized action representations (Lee et al., 2024; Pertsch et al., 2025), and many recent generalist vision-language-action policies (Kim et al., 2025; Black et al., 2024; Physical Intelligence et al., 2025; 2026).

Real-time inference for action chunking policies. A practical consequence of deploying large generative chunk policies in real time is that launching inference becomes part of the control problem. Each query to π_{chunk} can take tens to hundreds of milliseconds, which is non-negligible relative to high-frequency robot control loops (Black et al., 2025a; Kim et al., 2025; Prasad et al., 2024). Action chunking already lowers the effective inference rate by executing several actions per model call, and recent methods further adapt π_{chunk} to delay by incorporating delayed execution into training or by modifying inference-time guidance (Black et al., 2025b; Tang et al., 2025; Liao et al., 2025; Liu et al., 2026b). These improvements, however, still focus on the action generator itself. They do not remove the need to decide when the next generation call should be launched. Many systems therefore use fixed or heuristic re-query schedules, for example by executing the full chunk open-loop, re-querying every k steps, or overlapping inference with execution so the robot keeps acting while the next chunk is being generated (Chi et al., 2025; Black et al., 2025a; Ye et al., 2026). Lightweight execution verifiers can also trigger replanning when current observations indicate execution drift (Wang et al., 2026c). Our work learns Wait/Launch decisions through a latency-aware decision-stage SMDP around a fixed action generator.

Meta-control and delayed MDPs. A broad line of work studies MDPs and reinforcement learning with exogenous delays, including delayed state information or observations (Altman & Nain, 1992; Walsh et al., 2007; Agarwal & Aggarwal, 2021; Chen et al., 2024; Wu et al., 2024), delayed action execution (Katsikopoulos & Engelbrecht, 2003; Schuitema et al., 2010; Firoiu et al., 2018; Derman et al., 2021), and random action-observation delays in real-time reinforcement learning (Ramstedt & Pal, 2019; Bouteiller et al., 2021; Xia et al., 2025). In these formulations, delay is environment-imposed: actions, observations or feedback may arrive late, and the policy must compensate without controlling when the delay occurs. By contrast, in our setting, delay is incurred through a controllable launch decision. The controller chooses when to launch π_{chunk}, and the delayed object is the availability of the next action chunk rather than a primitive action or observation.

3 Action Generation and Meta Control under Inference Delay

We consider a finite-horizon physical MDP

$$\mathcal{M}_{\text{physical}} = (\mathcal{S}, \mathcal{A}, H, \{P_h\}_{h=1}^{H}, \{r_h\}_{h=1}^{H}),$$

where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the primitive action space, P_h is the transition kernel, r_h is the reward, and H is the horizon. At time h, the system has a physical state s_h and a time-indexed action buffer

$$B_h : \{1, 2, \dots, H\} \to \mathcal{A} \cup \{\perp\}.$$

We denote $B_h[t]$ as the primitive action to be executed at physical time t, and $\perp$ denotes an empty buffer entry. The action generation policy π_{chunk} maps the current state-buffer context to a length-L action chunk:

$$z_h = a_{h:h+L-1} := (a_h, \dots, a_{h+L-1}) \sim \pi_{\text{chunk}}(\cdot \mid s_h, B_h).$$

This definition offers great flexibility in designing the action chunking policy, allowing π_{chunk} to leverage available action information in B_h (Black et al., 2025b; Randazzo, 2026) or incorporate inference-time guidance (Black et al., 2025a; Liu et al., 2026b). The inference delay is a non-negative integer-valued random variable $\Delta_h(s_h)$ that is state dependent and supported on an interval $[\Delta_{\min}, \Delta_{\max}]$. An inference initiated at time h will return at time $h + \Delta_h(s_h)$.

Interaction Protocol The interaction protocol centers on action-buffer operations. We define an augmented meta state as $x_h = (s_h, B_h, \lambda_h, m_h) \in \mathcal{X}$, where λ_h is the launch time of the most recent inference and m_h is the pending-inference record:

$$m_h = \begin{cases} \varnothing, & \text{if no inference is pending,} \\ (\Delta_{\lambda_h}(s_{\lambda_h}), z_{\lambda_h}), & \text{if an inference launched at } \lambda_h \text{ is pending.} \end{cases}$$

The pending record m_h describes an inference in progress; its generated chunk z_{λ_h} becomes available to the controller only when inference returns. Throughout the paper, we assume that there can be at most one active inference, and that an active inference call cannot be interrupted, cancelled, or replaced before it returns.

Action-buffer update

At the beginning of step h, the controller first processes any pending inference that is scheduled to return at time h: the returned chunk is merged into the action buffer by a fixed buffer-update rule and the pending record is cleared. After the meta control decision below and any possible same-step return are processed, an action $a_h = B_h[h]$ is executed with reward $r_h(s_h, a_h)$, and the physical state evolves according to $P_h(\cdot \mid s_h, a_h)$.

Meta-control decision

If no inference is pending after arrival, the meta control policy π_{meta} chooses a feasible meta action. In the binary launch-timing case, the action is $u_h \in \mathcal{U}_h(x_h) \subseteq \{\text{wait}, \text{launch}\}$.

The initial buffer and feasible meta actions ensure that each executed buffer entry is nonempty. The buffer-update rule remains fixed while the meta policy chooses launch times.

Physical-Time Meta-Control MDP and Decision-Stage Semi-MDP For a fixed action generator π_{chunk}, the meta control is a finite-horizon MDP over the augmented state x_h. The transition kernel and reward are induced by the physical MDP dynamics, the buffer update rule, the delay distribution, and π_{chunk}. See Figure 1 for a demonstration. The formal transition kernel is given in Appendix A.1. The expected accumulated reward depends on both policies π_{chunk} and π_{meta}. At physical time h, given a meta state x, define

$$V_h^{\pi_{\text{chunk}},\pi_{\text{meta}}}(x) = \mathbb{E}^{\pi_{\text{chunk}},\pi_{\text{meta}}}\left[\sum_{t=h}^{H} r_t(s_t, a_t) \,\middle|\, x_h = x\right].$$

We typically freeze the action generator π_{chunk}, and therefore use the shorthand $V_h^{\pi_{\text{meta}}}(x)$. For

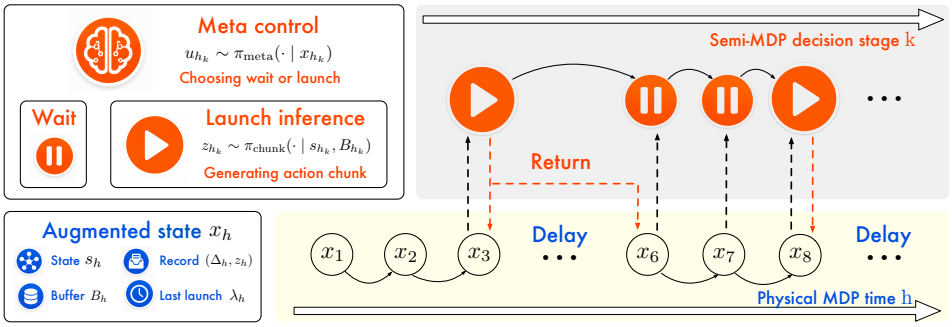

Figure 1: Physical meta-control MDP and its decision-stage semi-MDP under inference delay.

learning π_{meta}, it is useful to view the meta control MDP only at decision stages. A decision stage is a post-arrival physical time at which no inference is pending and a feasible meta action is selected. We use k to index these meta decision stages and h_k for the corresponding physical time. If the controller waits, then $h_{k+1} = h_k + 1$; if inference launches, then $h_{k+1} = h_k + \max\{1, \Delta_{h_k}(s_{h_k})\}$. During the inference pending interval, there is no new meta decision, and the controller executes buffered primitive actions until the inference result returns. We therefore collapse the launch interval into one transition from the current meta state x_{h_k} to the next no-pending meta state $x_{h_{k+1}}$, with reward $R_k = \sum_{t=h_k}^{h_{k+1}-1} r_t(s_t, a_t)$. To this end, we cast the meta control MDP into a Semi-MDP (SMDP, Sutton et al. (1999); Precup (2000)), where each decision stage corresponds to a variable-length physical-time interval. This SMDP view is useful because its transition law integrates over physical dynamics, stochastic delay, and chunk-generation randomness during each launch interval; the resulting Q-function can therefore compare launching now against waiting for more informative states. Section 5 defines its Q-function and learns π_{meta} via fitted Q-iteration.

4 Why Meta Control Matters

This section articulates the importance of meta control. We show through examples that with a fixed action generator π_{chunk}, different choices of meta-control policy π_{meta} result in substantial separations

in the expected accumulated reward. Section 4.1 demonstrates a non-monotonic dependence of value on inference timing and Section 4.2 shows the benefit of appropriate inference timing in a safety-critical scenario. Specifically, following the design of Training-Time RTC (Black et al., 2025b), the examples in this section fix a protection length p: returned chunks overwrite their unexecuted buffer entries only after the first p steps from launch.

4.1 Non-monotonic Effect of Inference Timing

As mentioned, a common rule is to launch π_{chunk} whenever a new inference call is feasible. This rule maximizes the chance that a returned chunk becomes active early, but during the inference, it blocks other inference calls. Therefore, bad launch timing may cause the controller to miss critical decision-relevant state information that will be observed. The following example shows that waiting strictly improves value, while excessive waiting strictly decreases value.

For a waiting time $w \in \mathbb{N}$, let $\pi^{(w)}_{\text{meta}}$ be the meta policy that waits w steps before the next launch and then chooses future launch times optimally. We write $V^{(w)}_h(x_h) = V^{\pi^{(w)}_{\text{meta}}}_h(x_h)$, and use $V^{(w)}_1(x_1)$ for the value at the initial state. In particular, $w = 0$ is immediate inference launch. The following proposition shows the non-monotonic dependence of $V^{(w)}_1$ on w.

Proposition 1 (Non-monotonic effect of launch timing)**.** *Fix any horizon $H \geq 16$. There exists a physical MDP instance such that, with chunk length $L = 3$, constant inference delay $\Delta = 1$, fixed protection length $p = 2$, and a fixed action generation policy π_{chunk}, waiting increases the value:*

$$V^{(2)}_1(x_1) - V^{(0)}_1(x_1) \geq H/5.$$

Moreover, further increasing the wait decreases the value: for every integer $2 \leq w \leq H - 3$, it holds that

$$V^{(2)}_1(x_1) - V^{(w)}_1(x_1) = w - 2.$$

The proof and explicit MDP construction are provided in Appendix B. The constructed MDP introduces branch uncertainty through the transition at $h = 2$, whose outcome is observed at $h = 3$. Before such a state is observed, an action chunk must choose *blindly* between two action segments: one that exploits a good state and one that recovers from an error state. Waiting until $w = 2$ lets π_{chunk} condition on the revealed state and select the correct action segment. This "value-of-information" effect makes $V^{(2)}_1(x_1)$ strictly larger than $V^{(0)}_1(x_1)$, where immediate launch commits to a chunk before the uncertainty is resolved. The second part of Proposition 1 shows the opposite effect. Once the state at time $h = 3$ is known, additional waiting only causes value loss.

4.2 Hidden-danger GridWorld Example

We further study the inference timing effect in a hidden-danger GridWorld (Towers et al., 2024). The agent starts near the lower-left corner and must reach the upper-right destination in Figure 2. The highlighted 3×3 danger zones are lethal, but they are only visible when the agent is within 3 blocks. The fixed action generator π_{chunk} is a scripted local planner that outputs an $L = 8$ action chunk: it follows a preset nominal road by default, treats unrevealed danger zones as free space, and, once a danger zone is revealed, takes a short detour around it before rejoining the nominal road.

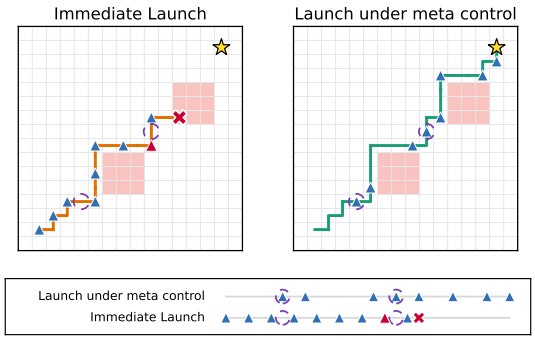

Figure 2: A hidden-danger rollout with fixed π_{chunk}. Purple circles mark first-reveal times, blue triangles mark launches, the red triangle marks a premature immediate launch, and the red cross marks danger entry.

We set a constant delay $\Delta = 2$ and protection length $p = 2$. Therefore, a newly launched chunk affects execution after $\max\{\Delta, p\} = 2$ steps. The immediate launch rule, which launches whenever inference is free, can therefore request a chunk before a nearby danger is revealed; waiting until the reveal lets the same π_{chunk} generate an avoidance chunk. Figure 2 shows

such a rollout: the immediate launch policy in the left panel queries too early, while π_{meta}, learned with Algorithm 1 introduced below, waits for danger-zone information in the right panel. This rollout illustrates that the learned policy can time inference to use newly revealed information. Appendix D.1 gives the full setting and quantitative results.

Consider the critical state marked by the red triangle in the left panel of Figure 2, where inference is free but the nearby danger has not yet been revealed. At this state, we compute the value functions defined in Section 4.1 for different waiting times before launch:

$$V_h^{(0)}(x_h) = -0.80, \quad V_h^{(1)}(x_h) = 1.60, \quad V_h^{(2)}(x_h) = -0.80.$$

Here the reward is positive for moving toward and reaching the goal, and negative for entering a danger zone. Thus $w = 1$ is the best option, which mirrors the non-monotonic effect in Proposition 1.

5 Learning a Meta-Control Policy

Section 4 shows that launch timing can change the value of a fixed action generator. The decision-stage SMDP in Section 3 turns each Wait or Launch choice into a transition to the next meta-control decision, accumulating rewards over the intervening physical steps. This formulation allows standard RL methods to learn launch timing from physical-time trajectories. We develop the resulting meta-control policy via fitted Q-iteration in Section 5.1 and analyze its performance in Section 5.2.

5.1 Decision-Stage Semi-MDP Fitted-Q Learning

We work with the decision-stage SMDP introduced in Section 3. Suppose we have N i.i.d. pre-collected physical-time trajectories $\{(x_{i,h}, r_{i,h}, u_{i,h})_{h=1}^{H}\}_{i=1}^{N}$. After preprocessing, we obtain $K_i \le H$ decision-stage SMDP transitions

$$\mathcal{D}^{(i)} = \{(h_{i,k}, x_{i,k}, u_{i,k}, R_{i,k}, h_{i,k+1}, x_{i,k+1})\}_{k=1}^{K_i}.$$

Here $h_{i,k}$ is the physical time of the k-th decision stage in trajectory i, $x_{i,k}$ is the corresponding post-arrival meta state, and $R_{i,k} = \sum_{t=h_{i,k}}^{h_{i,k+1}-1} r_{i,t}$ is the accumulated reward over the holding interval. We further write $\mathcal{D}_N = \{\mathcal{D}^{(i)}\}_{i=1}^{N}$.

Recall that k is the decision stage and h_k is the physical time. We denote by $Q_k^\dagger(h, x, u)$ the optimal SMDP Q-function in the decision-stage SMDP induced by the fixed π_{chunk}, and let $V_k^\dagger(h, x) = \max_{u \in \mathcal{U}_h(x)} Q_k^\dagger(h, x, u)$. For $k \le H$, the SMDP Bellman optimality equation is

$$Q_k^\dagger(h, x, u) = (\mathcal{T}_k^u V_{k+1}^\dagger)(h, x) := \mathbb{E}\Big[R_k + V_{k+1}^\dagger(h_{k+1}, x_{h_{k+1}}) \,\Big|\, h_k = h,\ x_{h_k} = x,\ u_{h_k} = u\Big],$$

with the terminal convention $Q_k^\dagger \equiv V_k^\dagger \equiv 0$ for all $k > H$. The operator $\mathcal{T}_k^u$ is known as the Bellman operator. Since the meta action is binary, Algorithm 1 estimates two action-specific Q-functions $Q_k^{\mathsf{wait}}(h, x)$ and $Q_k^{\mathsf{launch}}(h, x)$, with $Q_k(h, x, u) = Q_k^u(h, x)$. Algorithm 1 summarizes a fitted-Q iteration on the decision-stage tuples defined above.

Algorithm 1: Fitted Q Learning for the Decision-Stage SMDP

Require: Replay trajectories; Q-function classes $\{(\mathcal{F}_k^{\mathsf{wait}}, \mathcal{F}_k^{\mathsf{launch}})\}_{k=1}^{H}$.

Initialize $\widehat{Q}_{H+1} \equiv 0$.

for $k = H, H-1, \ldots, 1$ **do**

  $\mathcal{I}_k \leftarrow \{i \in \{1, \ldots, N\} : K_i \ge k\}$.

  for *each* $i \in \mathcal{I}_k$ **do**

    $Y_{i,k} \leftarrow R_{i,k} + \max_{u' \in \mathcal{U}_{h_{i,k+1}}(x_{i,k+1})} \widehat{Q}_{k+1}(h_{i,k+1}, x_{i,k+1}, u')$.

  Fit the two action-specific critics separately:

$$\widehat{Q}_k^u \in \operatorname{argmin}_{f \in \mathcal{F}_k^u} \sum_{i \in \mathcal{I}_k,\, u_{i,k} = u} \left(f(h_{i,k}, x_{i,k}) - Y_{i,k}\right)^2.$$

  Define $\widehat{Q}_k(h, x, u) \leftarrow \widehat{Q}_k^u(h, x)$ for $u \in \{\mathsf{wait}, \mathsf{launch}\}$.

return $\widehat{\pi}_{\text{meta},k}(h, x) = \operatorname{argmax}_{u \in \mathcal{U}_h(x)} \widehat{Q}_k(h, x, u)$ *as the greedy policy.*

Lines 3-5 are the SMDP-specific part of data pre-processing: samples are grouped by decision index k. For each k and $u \in \{\mathsf{wait}, \mathsf{launch}\}$, the effective sample size is $n_{k,u} = |\{i \in \{1, \dots, N\} : K_i \geq k,\ u_{i,k} = u\}|$. Since k advances only at no-pending meta decisions, it is not evenly spaced in physical time. As a result, some decision indices and physical-time regions may be well represented, while others may need additional exploration. Our analysis in Section 5.2 records this dependence through $n_{k,u}$. Line 6 estimates the wait and launch Q-functions as two separate regression targets. In implementation, the critic classes $\mathcal{F}_k^{\mathsf{wait}}$ and $\mathcal{F}_k^{\mathsf{launch}}$ share most parameters. Line 7 uses a greedy policy, while alternatives can also be adopted.

We also remark that the offline replay data $\mathcal{D}_N$ is collected by rolling out an exploration behavior meta policy $\pi_{\mathrm{meta}}^{\mathrm{b}}$. Good coverage—the behavior policy visits the relevant no-pending decision contexts—is important to fitted Q estimation. To improve the coverage, Algorithm 1 may be embedded in an online loop: deploy the last learned meta policy (the softmax version), collect experience trajectories, append their decision-stage tuples to $\mathcal{D}_N$, and periodically rerun fitted Q-iteration so that coverage and the learned meta policy improve over time. For Kinetix and DOM, we fit on offline datasets collected by the behavior policies described in Appendix D.

5.2 Performance Sub-optimality Analysis

We analyze Algorithm 1 on a fixed offline dataset under standard assumptions of bounded rewards and approximate Bellman completeness (Assumptions 1 and 2 in Appendix C.1). The bound separates three sources of sub-optimality: expert chunk quality, action-generator imitation error, and fitted-Q estimation error.

Recall that for a given π_{chunk} and π_{meta}, we denote the value function by $V_1^{\pi_{\mathrm{chunk}}, \pi_{\mathrm{meta}}}$. The optimal value is denoted as $V_1^\star(x_1) = \sup_{(\pi_{\mathrm{chunk}}, \pi_{\mathrm{meta}})} V_1^{\pi_{\mathrm{chunk}}, \pi_{\mathrm{meta}}}(x_1)$, where the buffer-update rule, delay law, and feasible meta-action sets are held fixed. We denote $\pi_{\mathrm{chunk}}^\star$ and $\pi_{\mathrm{meta}}^\star$ as the optimal policies attaining $V_1^\star$. For the returned meta control policy $\widehat{\pi}_{\mathrm{meta}}$ from Algorithm 1, we define its sub-optimality gap by $\mathtt{SubOpt}(\pi_{\mathrm{chunk}}, \widehat{\pi}_{\mathrm{meta}}) = V_1^\star(x_1) - V_1^{\pi_{\mathrm{chunk}}, \widehat{\pi}_{\mathrm{meta}}}(x_1)$. The action policy π_{chunk}, obtained by imitating an expert policy $\pi_{\mathrm{chunk}}^{\mathrm{exp}}$ [1], remains fixed throughout Algorithm 1. For the fixed π_{chunk}, we also denote $\pi_{\mathrm{meta}}^\dagger \in \mathrm{argmax}_{\pi_{\mathrm{meta}}} V_1^{\pi_{\mathrm{chunk}}, \pi_{\mathrm{meta}}}(x_1)$. The sample trajectories $\mathcal{D}_N$ are collected under an unknown behavior meta control policy $\pi_{\mathrm{meta}}^{\mathrm{b}}$.

We write ϵ for the Bellman approximation error and $\mathcal{N}_k(\zeta)$ for a common L_∞ covering-number bound at radius $0 < \zeta \leq H$ for the critic classes at stages k and $k+1$ (Wainwright, 2019). The coefficient C_k measures visitation mismatch between replay data and the policies under evaluation. Appendix C.2 provides the formal definitions and supporting discussion.

The theorem below establishes an upper bound on $\mathtt{SubOpt}$.

Theorem 1 (Sub-optimality bound)**.** *Suppose Assumptions 1 and 2 hold. For an absolute constant $c > 0$, with probability at least $1 - \delta$, the following bound holds whenever $n_{k,u} > 0$ for all k and u:*

$$\mathtt{SubOpt}(\pi_{\mathrm{chunk}}, \widehat{\pi}_{\mathrm{meta}}) \leq c(\mathcal{B}_{\mathrm{exp}} + \mathcal{B}_{\mathrm{imit}} + \mathcal{B}_{\mathrm{est}}),$$

where

$$\mathcal{B}_{\mathrm{exp}} = H \sum_{h=1}^H \mathbb{E}^{\pi_{\mathrm{chunk}}^\star, \pi_{\mathrm{meta}}^\star} \left[\mathsf{d}_{\mathrm{TV}} \left(\pi_{\mathrm{chunk},h}^\star(\cdot \mid s_h, B_h), \pi_{\mathrm{chunk},h}^{\mathrm{exp}}(\cdot \mid s_h, B_h) \right) \right],$$

$$\mathcal{B}_{\mathrm{imit}} = H \sum_{h=1}^H \mathbb{E}^{\pi_{\mathrm{chunk}}^{\mathrm{exp}}, \pi_{\mathrm{meta}}^\star} \left[\mathsf{d}_{\mathrm{TV}} \left(\pi_{\mathrm{chunk},h}^{\mathrm{exp}}(\cdot \mid s_h, B_h), \pi_{\mathrm{chunk},h}(\cdot \mid s_h, B_h) \right) \right],$$

$$\mathcal{B}_{\mathrm{est}} = \sum_{k=1}^H \kappa_k \left[\epsilon + \inf_{0 < \zeta \leq H} \left(H^2 \max_{u \in \{\mathsf{wait}, \mathsf{launch}\}} \frac{\log \mathcal{N}_k(\zeta) + \log(4H/\delta)}{n_{k,u}} + H\zeta \right)^{1/2} \right],$$

with $\kappa_k = C_k(\rho_k^\dagger, \rho_k^{\mathrm{b}}) + C_k(\widehat{\rho}_k, \rho_k^{\mathrm{b}})$. Here $\rho_k^\dagger, \rho_k^{\mathrm{b}}, \widehat{\rho}_k$ are visitation measures over tuples (h, x, u) induced by $\pi_{\mathrm{meta}}^\dagger$, $\pi_{\mathrm{meta}}^{\mathrm{b}}$, and $\widehat{\pi}_{\mathrm{meta}}$, respectively.

The proof is provided in Appendix C.3. We observe three components in the upper bound. The term $\mathcal{B}_{\mathrm{exp}}$ measures the mismatch between $\pi_{\mathrm{chunk}}^{\mathrm{exp}}$ and $\pi_{\mathrm{chunk}}^\star$, which can be small, e.g., at low latency. The term $\mathcal{B}_{\mathrm{imit}}$ measures how closely π_{chunk} approximates $\pi_{\mathrm{chunk}}^{\mathrm{exp}}$. The term $\mathcal{B}_{\mathrm{est}}$ is the statistical error from fitted Q-iteration. It contains the Bellman completeness error ϵ, the finite-sample

[1] This chunk policy is induced by the instantaneous expert π^{exp}. Appendix C.2 gives one such construction and defines the covering number used below.

regression error, and the distribution shift coefficients. Better coverage reduces the estimation term by simultaneously increasing sample counts and decreasing the mismatch between training and testing.

6 EXPERIMENTS

We evaluate launch-time meta control on Kinetix and then scale to vision-language-action (VLA) control on the dynamic manipulation benchmark DOM (Xie et al., 2026). Each comparison keeps the action generator π_{chunk} fixed and adds π_{meta} as a timing controller, testing the same formulation across different chunk policies. We report success rate (SR, %, ↑) and mean action-model calls per episode (Calls, ↓), including prefill. Across both benchmarks, π_{meta} improves SR over the corresponding action generator under every evaluated delay law.

6.1 KINETIX BENCHMARK

We use the 12-task Kinetix suite from RTC (Black et al., 2025a) with a Training-Time RTC backbone (Black et al., 2025b), denoted TrainingRTC. We evaluate four fixed delays, $\Delta \in \{0, 1, 2, 3\}$, and four stochastic laws over the same support. We compare learned timing with TrainingRTC, which launches whenever inference is free and launch is feasible. All experimental details and heuristic comparisons are in Appendices D.2–D.4.

Table 1: TrainingRTC SR (%, ↑) and Calls (↓) per episode, including prefill. Bold marks higher SR.

Metric	Method	Point_0	Point_1	Point_2	Point_3	Uniform	Geo	RevGeo	Binom	Avg.
SR ↑	TrainingRTC w/ π_{meta}	**92.74**	**89.75**	**87.15**	**84.72**	**85.84**	**86.54**	**84.88**	**85.78**	**87.17**
	TrainingRTC	90.90	88.53	85.90	83.78	84.85	85.59	84.35	85.26	86.15
Calls ↓	TrainingRTC w/ π_{meta}	60.40	57.57	38.97	29.73	45.09	55.30	35.61	45.53	46.02
	TrainingRTC	87.42	95.21	50.25	34.90	58.26	79.16	44.29	62.46	63.99

Results. Meta control improves SR under all eight delay laws, raising the mean from 86.15% to 87.17% (Table 1). At the same time, Calls fall from 63.99 to 46.02 per episode, a 28.08% reduction. These gains across fixed and stochastic delays show that learned launch timing improves success while reducing calls to the same generator.

Task difficulty and backbone. Table 2 ranks task–delay pairs by independent baseline SR evaluations for each backbone: Q1 contains the hardest 25% and Q2 the next hardest 25%. TrainingRTC's relative SR gain reaches 6.15% on the hardest quartile, compared with 0.59% on Q2 and 1.19% overall.

Table 2: SR (%, ↑) by difficulty. Q1 is the hardest quartile.

Method	Overall ↑	Q1 ↑	Q2 ↑
TrainingRTC w/ π_{meta}	**87.17**	**67.46**	**88.01**
TrainingRTC	86.15	63.56	87.49
FutureRTC w/ π_{meta}	**89.52**	**74.23**	**90.55**
FutureRTC	88.98	72.49	90.31

FutureRTC (Jiang et al., 2026) conditions action generation on predicted execution-time context. Learned timing improves its mean SR from 88.98% to 89.52%. The relative gain is 2.40% in Q1 versus 0.61% overall, following the same pattern as TrainingRTC. The benefit on harder tasks thus extends to a stronger π_{chunk}, supporting meta control across chunk policies. Appendix D.3 gives the ensemble and aggregation details.

6.2 VISION-LANGUAGE-ACTION EVALUATION ON DOM

We use the released DynamicVLA policy on DOM (Xie et al., 2026). DynamicVLA launches whenever inference is free; we compare this default schedule with the same policy under learned timing. Closed-loop Reactivity (CR) tests tracking and grasping objects moving at different speeds; Dynamic Adaptation (DA) adds abrupt motion changes, such as collision-induced deflections. Both require placing the grasped target at the instructed destination.

To evaluate timing under real-time inference constraints, we profile DynamicVLA on an RTX A6000 at a control frequency of 25 Hz (40 ms per tick). The measured latency averages approximately six control ticks, motivating Point_6 and $\text{Uniform}_{4:8}$ as fixed and variable delay settings. We also

evaluate the empirical latency distribution over 6–10 ticks to capture the measured runtime variation. Appendix D.5 gives all experimental details and heuristic comparisons.

Table 3: DynamicVLA on DOM: SR (%, ↑) and Calls (↓) per episode, including prefill. Bold marks higher SR.

		$\text{Uniform}_{4:8}$		Point_6		Empirical	
Task	Method	SR ↑	Calls ↓	SR ↑	Calls ↓	SR ↑	Calls ↓
CR	DynamicVLA w/ π_{meta}	**50.50**	25.30	**50.00**	24.34	**50.25**	23.99
CR	DynamicVLA	40.75	37.76	43.50	35.87	44.00	34.28
DA	DynamicVLA w/ π_{meta}	**42.00**	33.35	**38.25**	33.21	**40.25**	33.62
DA	DynamicVLA	28.50	40.05	24.25	40.17	25.00	40.67

Results. Meta control improves SR over DynamicVLA in all six settings (Table 3), with relative gains of 14.20–23.93% on CR and 47.37–61.00% on DA. For example, DA SR under $\text{Uniform}_{4:8}$ rises from 28.50% to 42.00%, a 47.37% relative increase. Calls also decrease under every delay law: by 30.02–33.00% on CR and 16.73–17.33% on DA.

DA has lower SR than CR for both policies and requires adaptation to abrupt motion changes. A call launched just before a direction change uses pre-change observations and remains pending as new information arrives; learned waiting can shift inference to a more informative observation. The larger DA gains, despite smaller call reductions than on CR, are consistent with launch timing being more consequential in this setting. This pattern complements the Kinetix quartile results: the clearest gains appear in difficult settings, with a particular advantage for abrupt adaptation on DOM.

Execution cost. Table 4 breaks down execution on an RTX A6000. Pre-fusion acquisition uses the frozen DynamicVLA's existing image encoder, language embeddings, and state projection. We pool these features for the meta policy, with no further training of the VLA. The meta policy then chooses Wait or Launch. A Launch continues the standard DynamicVLA computation: filling the key–value (KV) cache and running ten flow steps to generate the chunk. The Wait path stops after the timing decision.

Table 4: DOM execution latency (ms). Wait and Launch are measured as complete paths; P95 is the 95th-percentile latency.

Stage	Median	P95
Pre-fusion acquisition	18.22	18.67
Meta decision	0.88	1.02
Wait subtotal	18.76	19.18
KV fill	20.95	21.82
Flow generation (10 steps)	193.51	203.92
Launch total	234.12	241.93

The meta network has 1.21M parameters, 0.28% of the 430.19M-parameter DynamicVLA backbone. Its decision takes 0.88 ms, 0.37% of the 234.12 ms Launch path. The complete Wait path takes 18.76 ms (median; 8.01% of a Launch) and 19.18 ms at P95, both below the 40 ms control period. Flow generation dominates launch latency, making observation checks inexpensive while the controller reduces action-model calls (Table 3). Appendix D.5 reports prefix counts and the measurement protocol.

7 Conclusion

Launch timing can substantially affect control performance even when the action generator is fixed. We formulate this timing decision as a meta-control problem. The induced decision-stage SMDP yields a model-free fitted-Q procedure for learning launch timing rules. Our theory first shows why waiting can help when newly revealed information outweighs the lost activation window, then provides a sub-optimality guarantee that separates expert chunk quality, action-generator imitation error, and fitted-Q estimation error.

Across Kinetix and DOM, learned timing improves success while reducing calls to fixed action generators: TrainingRTC uses 28.08% fewer calls on Kinetix, and DOM Dynamic Adaptation gains 47.37–61.00% in relative success rate. The larger improvements on hard Kinetix tasks and abrupt DOM motion changes support learning when to acquire a new action chunk as a useful control decision.

AI USE STATEMENT

Generative AI tools were used to discuss manuscript drafts and experimental design, review experiment code and result interpretations, and draft and polish portions of the text. The authors reviewed the AI-assisted material and take responsibility for the paper's claims, results, and final content.

REPRODUCIBILITY STATEMENT

The formal assumptions and proofs are provided in Appendices A–C. Algorithm 1 and Appendix D describe the learning procedure, implementation choices, and evaluation protocols needed to implement the method and reproduce the reported experiments. We plan to release the code in the future.

REFERENCES

Mridul Agarwal and Vaneet Aggarwal. Blind decision making: Reinforcement learning with delayed observations. *Pattern Recognition Letters*, 150:176–182, 2021.

Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? *arXiv preprint arXiv:2211.15657*, 2022.

Eitan Altman and Philippe Nain. Closed-loop control with delayed information. *ACM sigmetrics performance evaluation review*, 20(1):193–204, 1992.

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, Manuel Y Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. *arXiv preprint arXiv:2506.07339*, 2025a.

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

Yann Bouteiller, Simon Ramstedt, Giovanni Beltrame, Christopher Pal, and Jonathan Binas. Reinforcement learning with random delays. In *International conference on learning representations*, 2021.

Minshuo Chen, Haoming Jiang, Wenjing Liao, and Tuo Zhao. Efficient approximation of deep relu networks for functions on low dimensional manifolds. *Advances in neural information processing systems*, 32, 2019.

Minshuo Chen, Jie Meng, Yu Bai, Yinyu Ye, H Vincent Poor, and Mengdi Wang. Efficient reinforcement learning with impaired observability: Learning to act with delayed and missing state observations. *IEEE Transactions on Information Theory*, 70(10):7251–7272, 2024.

Ching-An Cheng, Tengyang Xie, Nan Jiang, and Alekh Agarwal. Adversarially trained actor critic for offline reinforcement learning. In *International Conference on Machine Learning*, pp. 3852–3878. PMLR, 2022.

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. doi: 10.1177/02783649241273668.

Esther Derman, Gal Dalal, and Shie Mannor. Acting in delayed environments with non-stationary markov policies. *arXiv preprint arXiv:2101.11992*, 2021.

Dennis Elbrächter, Dmytro Perekrestenko, Philipp Grohs, and Helmut Bölcskei. Deep neural network approximation theory. *arXiv preprint arXiv:1901.02220*, 2021.

Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. A theoretical analysis of deep q-learning. In *Proceedings of the 2nd Annual Conference on Learning for Dynamics and Control*, volume 120 of *Proceedings of Machine Learning Research*, pp. 486–489. PMLR, 2020.

Vlad Firoiu, Tina Ju, and Josh Tenenbaum. At human speed: Deep reinforcement learning with action delay. *arXiv preprint arXiv:1810.07286*, 2018.

Hengyu Fu, Zhuoran Yang, Mengdi Wang, and Minshuo Chen. Unveil conditional diffusion models with classifier-free guidance: A sharp statistical theory. *arXiv preprint arXiv:2403.11968*, 2024.

Niklas Funk, Julen Urain, Joao Carvalho, Vignesh Prasad, Georgia Chalvatzaki, and Jan Peters. Actionflow: Equivariant, accurate, and efficient policies with spatially symmetric flow matching. *arXiv preprint arXiv:2409.04576*, 2024.

Abraham George and Amir Barati Farimani. One act play: Single demonstration behavior cloning with action chunking transformers. *arXiv preprint arXiv:2309.10175*, 2023.

Zhi Hou, Tianyi Zhang, Yuwen Xiong, Haonan Duan, Hengjun Pu, Ronglei Tong, Chengyang Zhao, Xizhou Zhu, Yu Qiao, Jifeng Dai, et al. Dita: Scaling diffusion transformer for generalist vision-language-action policy. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7686–7697, 2025.

Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

Hai Jiang, Yixian Zou, Binbin Liang, Boqian Liu, Fanman Meng, and Shuaicheng Liu. FuturERTC: Real-time robot execution with anticipatory-conditioned action chunking. *arXiv preprint arXiv:2607.24008*, 2026. URL https://arxiv.org/abs/2607.24008.

Sunshine Jiang, Xiaolin Fang, Nicholas Roy, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Siddharth Ancha. Streaming flow policy: Simplifying diffusion/flow-matching policies by treating action trajectories as flow trajectories. *arXiv preprint arXiv:2505.21851*, 2025.

Ying Jin, Zhuoran Yang, and Zhaoran Wang. Is pessimism provably efficient for offline rl? In *International conference on machine learning*, pp. 5084–5096. PMLR, 2021.

Konstantinos V Katsikopoulos and Sascha E Engelbrecht. Markov decision processes with delays and asynchronous cost collection. *IEEE transactions on automatic control*, 48(4):568–574, 2003.

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

Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, Ming-Yu Liu, Chelsea Finn, and Jinwei Gu. Cosmos Policy: Fine-tuning video models for visuomotor control and planning. *arXiv preprint arXiv:2601.16163*, 2026.

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

Ye Li, Huanan Liu, Kangye Ji, Yuan Meng, Jiajun Fan, Yuansong Wang, Shiyu Qin, Chenglei Wu, Shu-Tao Xia, and Zhi Wang. ElegantVLA: Learning when to think for efficient vision-language-action models. *arXiv preprint arXiv:2605.29438*, 2026.

Yuanchang Liang, Xiaobo Wang, Kai Wang, Shuo Wang, Xiaojiang Peng, Haoyu Chen, David Kim Huat Chua, and Prahlad Vadakkepat. Adaptive action chunking at inference-time for vision-language-action models. *arXiv preprint arXiv:2604.04161*, 2026.

Aileen Liao, Dong-Ki Kim, Max Olan Smith, Ali-akbar Agha-mohammadi, and Shayegan Omidshafiei. Delay-aware diffusion policy: Bridging the observation-execution gap in dynamic tasks. *arXiv preprint arXiv:2512.07697*, 2025.

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

Jun Liu, Pu Zhao, Zhenglun Kong, Xuan Shen, Peiyan Dong, Fan Yang, Lin Cui, Hao Tang, Geng Yuan, Wei Niu, et al. When should a robot think? resource-aware reasoning via reinforcement learning for embodied robotic decision-making. *arXiv preprint arXiv:2603.16673*, 2026a.

Yufeng Liu, Hang Yu, Juntu Zhao, Bocheng Li, Di Zhang, Mingzhu Li, Wenxuan Wu, Yingdong Hu, Junyuan Xie, Junliang Guo, Dequan Wang, and Yang Gao. Learning native continuation for action chunking flow policies. *arXiv preprint arXiv:2602.12978*, 2026b.

Rémi Munos and Csaba Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9(27):815–857, 2008.

Junnan Nie, Jiayi Li, Jiachen Zhang, Junyi Lao, Chenghao Liu, Tianle Zhang, Liang Lin, and Songfang Huang. PACE: Phase-aware chunk execution for robot policies with action chunking. *arXiv preprint arXiv:2606.00537*, 2026.

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.

Kazusato Oko, Shunta Akiyama, and Taiji Suzuki. Diffusion models are minimax optimal distribution estimators. In *International Conference on Machine Learning*, pp. 26517–26582. PMLR, 2023.

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, 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.

Physical Intelligence, Bo Ai, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Greg Balke, Kevin Black, George Bokinsky, Shihao Cao, Thomas Charbonnier, et al. $\pi_{0.7}$: a steerable generalist robotic foundation model with emergent capabilities. *arXiv preprint arXiv:2604.15483*, 2026.

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

Doina Precup. *Temporal abstraction in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 2000.

Simon Ramstedt and Chris Pal. Real-time reinforcement learning. *Advances in neural information processing systems*, 32, 2019.

Giacomo Randazzo. ACTSmooth: Smooth ACT policies on the SO101 arm. https://github.com/giacomoran/lerobot-policy-act-smooth, 2026. Blog post: https://www.giacomoran.com/blog/act-smooth/.

Johannes Schmidt-Hieber. Nonparametric regression using deep neural networks with ReLU activation function. *The Annals of Statistics*, 48(4):1875–1897, 2020.

Erik Schuitema, Lucian Buşoniu, Robert Babuška, and Pieter Jonker. Control delay in reinforcement learning for real-time dynamic systems: A memoryless approach. In *2010 IEEE/RSJ international conference on intelligent robots and systems*, pp. 3226–3231. IEEE, 2010.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

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.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

Martin J. Wainwright. *High-dimensional statistics: A non-asymptotic viewpoint*, volume 48 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, 2019. doi: 10.1017/9781108627771.

Thomas J Walsh, Ali Nouri, Lihong Li, and Michael L Littman. Planning and learning in environments with delayed feedback. In *European Conference on Machine Learning*, pp. 442–453. Springer, 2007.

Haoxuan Wang, Gengyu Zhang, Yan Yan, Ramana Rao Kompella, and Gaowen Liu. VLA knows its limits: Adaptive execution horizons for robot policies. *arXiv preprint arXiv:2602.21445*, 2026a.

Haoxuan Wang, Gengyu Zhang, Yan Yan, Yuzhang Shang, Ramana Rao Kompella, and Gaowen Liu. Real-time robot execution with masked action chunking. *arXiv preprint arXiv:2601.20130*, 2026b.

Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*, 2022.

Zihua Wang, Zhitao Lin, Ruibo Li, Yu Zhang, Xu Yang, Siya Mi, and Xiu-Shen Wei. Open-loop planning, closed-loop verification: Speculative verification for VLA. *arXiv preprint arXiv:2604.02965*, 2026c.

Qingyuan Wu, Simon S Zhan, Yixuan Wang, Yuhui Wang, Chung-Wei Lin, Chen Lv, Qi Zhu, and Chao Huang. Variational delayed policy optimization. *Advances in neural information processing systems*, 37:54330–54356, 2024.

Zhichao Wu, Junyin Ye, Zhilong Zhang, Yihao Sun, Haoxin Lin, Jiaheng Luo, Haoxiang Ren, Lei Yuan, and Yang Yu. Speedup patch: Learning a plug-and-play policy to accelerate embodied manipulation. *arXiv preprint arXiv:2603.20658*, 2026.

Bo Xia, Haoyuan Sun, Bo Yuan, Zhiheng Li, Bin Liang, and Xueqian Wang. A delay-robust method for enhanced real-time reinforcement learning. *Neural Networks*, 181:106769, 2025.

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. URL https://arxiv.org/abs/2601.22153.

Tengyang Xie, Ching-An Cheng, Nan Jiang, Paul Mineiro, and Alekh Agarwal. Bellman-consistent pessimism for offline reinforcement learning. *Advances in neural information processing systems*, 34:6683–6694, 2021.

Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural networks*, 94: 103–114, 2017.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. *arXiv preprint arXiv:2403.03954*, 2024.

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.

Tony Z Zhao, Jonathan Tompson, Danny Driess, Pete Florence, Kamyar Ghasemipour, Chelsea Finn, and Ayzaan Wahid. Aloha unleashed: A simple recipe for robot dexterity. *arXiv preprint arXiv:2410.13126*, 2024.

A Proofs and Details for Section 3

A.1 Meta MDP Transition Probability

This subsection gives the formal transition probability for the physical-time meta control MDP in Section 3. Throughout this subsection, all states x_h are post-arrival states: arrivals scheduled for time h have already been processed before the meta policy is evaluated. The transition kernel below maps the post-arrival state x_h to the post-arrival state x_{h+1}. In particular, x' in $K_h^{\pi_{\text{chunk}}}(dx' \mid x, u)$ is the post-arrival state at time $h+1$.

For a chunk z_t launched at step t, view z_t as indexed by its absolute support $\{t, \dots, t+L-1\}$. Let $\mathsf{Write}_h(B; z_t, t)$ be the buffer obtained by merging this chunk at its return time h. This is a fixed, measurable deterministic map of the displayed arguments; it leaves already executed entries unchanged. The associated arrival operator is

$$\mathsf{Arrive}_h(B, \lambda, m) := \begin{cases} \big(\mathsf{Write}_h(B; z, \lambda), \lambda, \varnothing\big), & m = (\ell, z) \text{ and } h = \lambda + \ell, \\ (B, \lambda, m), & \text{otherwise.} \end{cases}$$

When $m = \varnothing$, the value of λ is only a bookkeeping variable.

Let

$$\mathcal{Z}_h := \mathcal{A}^{\{h, h+1, \dots, h+L-1\}}, \qquad q_h(\ell \mid s) := \mathbb{P}(\Delta_h(s) = \ell), \qquad \ell \in \{\Delta_{\min}, \dots, \Delta_{\max}\}.$$

Fix the chunk generator $\pi_{\text{chunk}} = \{\pi_{\text{chunk},h}\}_{h=1}^{H}$, and define the controlled post-arrival physical-time kernel

$$K_h^{\pi_{\text{chunk}}}(dx' \mid x, u), \qquad x = (s, B, \lambda, m), \quad x' = (s', B', \lambda', m'),$$

where $u \in \{\mathsf{wait}, \mathsf{launch}\}$ is the meta action.

The feasible meta-action set is

$$\mathcal{U}_h(x) := \begin{cases} \mathcal{U}_h^0(x) \subseteq \{\mathsf{wait}, \mathsf{launch}\}, & m = \varnothing, \\ \{\mathsf{wait}\}, & m \neq \varnothing. \end{cases}$$

In the unconstrained binary launch-timing case, $\mathcal{U}_h^0(x) = \{\mathsf{wait}, \mathsf{launch}\}$. Additional non-idling or buffer-safety constraints can be encoded by removing infeasible actions from $\mathcal{U}_h^0(x)$. When $m \neq \varnothing$, the singleton action wait represents an automatic continuation step.

Wait transition. For $u = \mathsf{wait}$, no new inference is launched. The primitive action executed at step h is $B[h]$. After the physical transition, arrivals scheduled for time $h+1$ are processed:

$$(B_{\mathsf{wait}}, \lambda_{\mathsf{wait}}, m_{\mathsf{wait}}) = \mathsf{Arrive}_{h+1}(B, \lambda, m).$$

Thus

$$K_h^{\pi_{\text{chunk}}}(dx' \mid x, \mathsf{wait}) = P_h(ds' \mid s, B[h])\, \delta_{B_{\mathsf{wait}}}(dB')\, \delta_{\lambda_{\mathsf{wait}}}(d\lambda')\, \delta_{m_{\mathsf{wait}}}(dm'),$$

where δ_y denotes a Dirac mass at y.

Launch transition. For a feasible launch, sample

$$\ell \sim q_h(\cdot \mid s), \qquad z \sim \pi_{\text{chunk},h}(\cdot \mid s, B).$$

For each sampled pair (ℓ, z), first process any same-step return from this new launch:

$$(B^+_{\mathsf{launch}}, \lambda^+_{\mathsf{launch}}, m^+_{\mathsf{launch}}) = \mathsf{Arrive}_h(B, h, (\ell, z)).$$

This expression covers the no-delay case $\ell = 0$: the newly launched chunk returns in the same physical step, is written into the buffer according to the fixed buffer-update rule, and the pending record is immediately cleared. We nevertheless allow at most one meta decision per physical step.

The primitive action executed at step h is $B^+_{\mathsf{launch}}[h]$. After the physical transition, arrivals scheduled for time $h+1$ are processed:

$$(B_{\mathsf{launch}}, \lambda_{\mathsf{launch}}, m_{\mathsf{launch}}) = \mathsf{Arrive}_{h+1}(B^+_{\mathsf{launch}}, \lambda^+_{\mathsf{launch}}, m^+_{\mathsf{launch}}).$$

Here $B^+_{\mathsf{launch}}, \lambda^+_{\mathsf{launch}}, m^+_{\mathsf{launch}}$ and $B_{\mathsf{launch}}, \lambda_{\mathsf{launch}}, m_{\mathsf{launch}}$ depend on the sampled (ℓ, z), and we suppress this dependence for readability. Then

$$K_h^{\pi_{\text{chunk}}}(dx' \mid x, \mathsf{launch}) = \sum_{\ell=\Delta_{\min}}^{\Delta_{\max}} q_h(\ell \mid s) \int_{\mathcal{Z}_h} P_h(ds' \mid s, B^+_{\mathsf{launch}}[h]) \times \delta_{B_{\mathsf{launch}}}(dB')\, \delta_{\lambda_{\mathsf{launch}}}(d\lambda')\, \delta_{m_{\mathsf{launch}}}(dm')\, \pi_{\text{chunk},h}(dz \mid s, B).$$

The launch kernel is used only when $\mathsf{launch} \in \mathcal{U}_h(x)$.

Rewards and closed-loop kernel. The corresponding one-step expected rewards are

$$\rho_h^{\pi_{\text{chunk}}}(x, \mathsf{wait}) = r_h(s, B[h]),$$

and, when launching is feasible,

$$\rho_h^{\pi_{\text{chunk}}}(x, \mathsf{launch}) = \sum_{\ell=\Delta_{\min}}^{\Delta_{\max}} q_h(\ell \mid s) \int_{\mathcal{Z}_h} r_h(s, B^+_{\mathsf{launch}}[h])\, \pi_{\text{chunk},h}(dz \mid s, B).$$

Given a meta policy $\pi_{\text{meta}} = \{\pi_{\text{meta},h}\}_{h=1}^{H}$, we use the convention that $\pi_{\text{meta},h}(\mathsf{wait} \mid x) = 1$ whenever $m \neq \varnothing$. The closed-loop physical-time transition kernel is

$$K_h^{\pi_{\text{chunk}},\pi_{\text{meta}}}(dx' \mid x) = \sum_{u \in \mathcal{U}_h(x)} \pi_{\text{meta},h}(u \mid x)\, K_h^{\pi_{\text{chunk}}}(dx' \mid x, u),$$

and the closed-loop expected reward is

$$\rho_h^{\pi_{\text{chunk}},\pi_{\text{meta}}}(x) = \sum_{u \in \mathcal{U}_h(x)} \pi_{\text{meta},h}(u \mid x)\, \rho_h^{\pi_{\text{chunk}}}(x, u).$$

Connection to the decision-stage SMDP. The decision-stage SMDP in Section 3 is obtained by observing this same physical-time process at post-arrival states x_h where the meta policy is evaluated. In the unconstrained binary launch-timing case, decision stages are exactly the post-arrival no-pending times. If the meta action is wait, the next decision stage occurs at $h + 1$. If the meta action is launch and the sampled delay is ℓ, then the next decision stage occurs at

$$h + \max\{1, \ell\}.$$

For $\ell = 0$, the chunk returns in the same physical step, but the next decision stage is still $h + 1$, since we allow at most one meta decision per physical step. For $\ell \geq 1$, the next decision stage is the post-arrival return time $h + \ell$. The SMDP reward is the accumulated primitive reward over the holding interval:

$$R_k = \sum_{t=h_k}^{h_{k+1}-1} r_t(s_t, a_t).$$

A.2 Bellman Recursion for a Fixed Chunk Generator

We record the fixed-generator Bellman recursion as a consequence of the formulation, since it is useful for implementation.

Proposition 2 (Bellman recursion for a fixed chunk generator)**.** *Fix a chunk policy π_{chunk}. For each post-arrival augmented state x, let $\mathcal{U}_h(x)$ be the feasible meta-action set defined in Appendix A.1. Define*

$$\mathcal{V}_{H+1}^{\pi_{\text{chunk}}}(x) := 0, \qquad \mathcal{V}_h^{\pi_{\text{chunk}}}(x) := \sup_{\sigma_{h:H}} \mathbb{E}\Big[\sum_{t=h}^{H} \rho_t^{\pi_{\text{chunk}}}(X_t, \sigma_t(X_t)) \,|\, X_h = x\Big].$$

Then, for every stage h and augmented state x,

$$\mathcal{V}_h^{\pi_{\text{chunk}}}(x) = \max_{u \in \mathcal{U}_h(x)} \mathcal{Q}_h^{\pi_{\text{chunk}}}(x, u),$$

where

$$\mathcal{Q}_h^{\pi_{\text{chunk}}}(x, u) := \rho_h^{\pi_{\text{chunk}}}(x, u) + \int \mathcal{V}_{h+1}^{\pi_{\text{chunk}}}(x')\, K_h^{\pi_{\text{chunk}}}(dx' \,|\, x, u).$$

An optimal start rule is therefore given by

$$\sigma_h^{\star,\pi_{\text{chunk}}}(x) \in \operatorname*{argmax}_{u \in \mathcal{U}_h(x)} \mathcal{Q}_h^{\pi_{\text{chunk}}}(x, u),$$

with the convention that $\sigma_h^{\star,\pi_{\text{chunk}}}(x) = \mathsf{wait}$ whenever $\mathsf{launch} \notin \mathcal{U}_h(x)$.

Proof. Once π_{chunk} is fixed, the only control variable at stage h is whether to wait or launch, subject to feasibility. By construction of the augmented state $X_h = (S_h, B_h, \lambda_h, m_h)$, the future evolution of the delayed execution process depends on the past only through the current augmented state and the current wait/launch decision, with transition kernel $K_h^{\pi_{\text{chunk}}}$ and expected reward $\rho_h^{\pi_{\text{chunk}}}$. Therefore the controlled process is a finite-horizon MDP on the augmented state space with action set $\mathcal{U}_h(x)$. The displayed Bellman recursion is exactly the standard principle of optimality for that MDP. □

B PROOF FOR SECTION 4

Proof of Proposition 1. The proof is constructive. We define a finite-horizon physical MDP, whose state space is

$$\mathcal{S} = \{s_{\text{good}}, s_{\text{err}}, s_{\text{dead}}\},$$

and the action space is

$$\mathcal{A} = \{a_{\text{exp}}, a_{\text{neu}}, a_{\text{rec}}\},$$

corresponding to exploitation, neutral action, and recovery. The initial state is $s_1 = s_{\text{good}}$. The reward is

$$r_h(s, a) = \mathbb{1}\{s = s_{\text{good}},\ a = a_{\text{exp}}\}.$$

The state s_{dead} is absorbing and gives zero reward.

Let $0 \le q \le 3/5$ be a constant. At time $h = 2$, the transition from s_{good} is stochastic and independent of the action:

$$P_2(s_3 = s_{\text{good}} \mid s_2 = s_{\text{good}}, a_2) = q, \qquad P_2(s_3 = s_{\text{err}} \mid s_2 = s_{\text{good}}, a_2) = 1 - q.$$

All other transitions are deterministic and given by

	s_{good}	s_{err}	s_{dead}
a_{exp}	s_{good}	s_{dead}	s_{dead}
a_{neu}	s_{good}	s_{err}	s_{dead}
a_{rec}	s_{dead}	s_{good}	s_{dead}.

Thus exploitation is optimal in the good state, recovery is needed in the error state, and applying the wrong action sends the system to the dead state.

Under the buffer-update rule in Section 4.1, a chunk launched at time h returns at time $h + 1$ and can write only coordinate $h + 2$. The initial buffer B_1 is filled with neutral actions, so execution remains well-defined even when the meta controller waits. Consequently, to define π_{chunk}, we omit the first two actions in a generated chunk; they will never be executed. Before the state s_3 is observed, the fixed action generator commits blindly to exploitation:

$$\pi_{\text{chunk},1}\big(z_1[3] = a_{\text{exp}} \mid s_{\text{good}}, B_1\big) = 1,$$
$$\pi_{\text{chunk},2}\big(z_2[4] = a_{\text{exp}} \mid s_{\text{good}}, B_2\big) = 1,$$

After s_3 is observed, the remaining state transitions are deterministic. Let $\Phi_h(s_h, B_h)$ be the state at the beginning of time $h + 2$ obtained by executing the buffer actions $B_h[h]$ and $B_h[h + 1]$ from s_h. For $h \ge 3$, the policy deterministically chooses among exploitation, recovery, and neutral:

$$\pi_{\text{chunk},h}\big(z_h[h+2] = a_{\text{exp}} \mid s_h, B_h\big) = 1 \quad \text{if } \Phi_h(s_h, B_h) = s_{\text{good}},$$
$$\pi_{\text{chunk},h}\big(z_h[h+2] = a_{\text{rec}} \mid s_h, B_h\big) = 1 \quad \text{if } \Phi_h(s_h, B_h) = s_{\text{err}},$$
$$\pi_{\text{chunk},h}\big(z_h[h+2] = a_{\text{neu}} \mid s_h, B_h\big) = 1 \quad \text{if } \Phi_h(s_h, B_h) = s_{\text{dead}}.$$

We set the initial meta state $x_1 = \{s_{\text{good}}, B_1, \lambda_1 = -1, m_1 = \varnothing\}$, where $m_1 = \varnothing$ indicates that no inference is pending.

Value calculation For $V_1^{(0)}$, the first launch occurs at time $h = 1$ and writes $a_3 = a_{\text{exp}}$, before the outcome $s_3 \in \{s_{\text{good}}, s_{\text{err}}\}$ is known. No later launch can modify a_3. Hence, on the error branch ($s_3 = s_{\text{err}}$), the exploitation action sends the system to s_{dead}, regardless of how subsequent actions are chosen. On the good branch ($s_3 = s_{\text{good}}$), reward can be collected at every step $h = 3, \dots, H$. Therefore, we have

$$V_1^{(0)}(x_1) = q(H - 2).$$

Next for $V_1^{(2)}$, the first launch occurs at time $h = 3$, after the branch outcome has been observed. The first coordinate written by this returned chunk affects a_5. If $s_3 = s_{\text{good}}$, the neutral buffered actions preserve the good state until $h = 5$, and the policy exploits from stage 5 onward. This gives a value $H - 4$. If $s_3 = s_{\text{err}}$, the policy predicts recovery at $h = 5$ and then exploits from stage $h = 6$ onward. This gives a value $H - 5$. These continuations are feasible because, after each return, the meta controller can relaunch and π_{chunk} accounts for the committed actions already in the buffer. Both values are optimal because no policy whose first launch is at time $h = 3$ can affect execution before time $h = 5$. Hence it holds that

$$V_1^{(2)}(x_1) = q(H - 4) + (1 - q)(H - 5) = H - 5 + q.$$

Subtracting the immediate launch value gives the gap

$$V_1^{(2)}(x_1) - V_1^{(0)}(x_1) = H - 5 + q - q(H-2) = H - 5 + 3q - qH.$$

By the choice $q \le 3/5$, for $H \ge 16$, we have

$$V_1^{(2)}(x_1) - V_1^{(0)}(x_1) = H - 5 + 3q - qH \ge \frac{2H-16}{5} \ge \frac{H}{5},$$

which proves the first claim that waiting increases the value.

To prove the second claim, let $2 \le w \le H-3$, so that the first useful action lies within the horizon. The first launch occurs at stage $h = w+1$. The branch outcome was already observed at $h = 3$; additional waiting provides no new information but only executes buffered neutral actions. It shifts the first useful action prediction from stage $h = 5$ to stage $h = w+3$. Therefore

$$V_1^{(w)}(x_1) = q(H-w-2) + (1-q)(H-w-3) = H - w - 3 + q.$$

As a result, we have

$$V_1^{(2)}(x_1) - V_1^{(w)}(x_1) = w - 2.$$

This completes the proof. □

C Proofs and Details for Section 5

C.1 Assumptions and Supporting Discussion

We analyze Algorithm 1 on a fixed offline dataset. The bound separates three sources of suboptimality: the quality of the expert demonstrations when reused in the delayed system, imitation error of the action generator π_{chunk}, and the fitted-Q estimation error. We first impose some standard assumptions from theoretical reinforcement learning. Holding intervals are truncated at $H+1$, and all critic and continuation values vanish at terminal contexts.

Assumption 1 (Bounded reward)**.** *The reward r_h is bounded in $[0,1]$ for every $h \in \{1,\dots,H\}$. At physical time h, every critic $f \in \mathcal{F}_k^u$ is clipped to $[0, H-h+1]$, with value zero at terminal contexts.*

Clipping respects the maximum remaining accumulated reward in $\mathcal{M}_{\text{physical}}$. For a scalar network output q at a nonterminal context, it can be implemented as

$$q \mapsto \mathrm{ReLU}(q) - \mathrm{ReLU}\big(q - (H-h+1)\big), \qquad 1 \le h \le H.$$

Assumption 2 (Bellman completeness)**.** *There exists $\epsilon \ge 0$ such that, for every decision stage k, every $u \in \{\mathsf{wait}, \mathsf{launch}\}$, and every continuation value V satisfying $0 \le V(h,x) \le H-h+1$ at nonterminal contexts and $V = 0$ at terminal contexts, there exists $\widetilde{f} \in \mathcal{F}_k^u$ satisfying*

$$\|\widetilde{f} - \mathcal{T}_k^u V\|_\infty \le \epsilon.$$

These ranges are compatible with the SMDP Bellman backup: since $0 \le R_k \le h_{k+1} - h_k$,

$$\begin{aligned} 0 \le R_k + V(h_{k+1}, x_{h_{k+1}}) &\le (h_{k+1} - h_k) + (H - h_{k+1} + 1) \\ &= H - h_k + 1. \end{aligned}$$

Thus, $\mathcal{T}_k^u V$ lies in the clipping range of $\mathcal{F}_k^u$.

Bellman completeness is a standard assumption in theoretical reinforcement learning (Munos & Szepesvári, 2008; Fan et al., 2020; Xie et al., 2021). Here it requires each action-specific critic class $\mathcal{F}_k^u$ to approximate the backups of continuations satisfying the stated range condition. Under suitable uniform smoothness assumptions on these backups, sufficiently expressive neural network classes can make the approximation error small (Yarotsky, 2017; Elbrächter et al., 2021). Clipping to the above range does not increase the approximation error.

C.2 Expert Chunk Policy and Complexity Measures

Demonstrations are collected under an expert policy $\pi_h^{\text{exp}}(a_h \mid s_h)$, which predicts instantaneous actions. One induced chunk policy is the distribution of length-L expert rollouts starting from s_h:

$$\pi_{\text{chunk},h}^{\text{exp}}(\mathrm{d}z_h \mid s_h, B_h) = \int \left[\prod_{t=h}^{h+L-1} \pi_t^{\text{exp}}(\mathrm{d}z_h[t] \mid s_t)\, P_t(\mathrm{d}s_{t+1} \mid s_t, z_h[t]) \right],$$

where the integral marginalizes the physical states $s_{h+1:h+L}$. Beyond the horizon, the state is absorbing and the expert outputs a fixed padding action. This rollout construction does not depend on B_h. Other constructions can condition on the buffer, as illustrated by the chunk generators in Section 4 and Appendix D. The analysis below applies to any fixed choice of the induced expert chunk policy.

The covering number is a classical concept in statistical learning. Given a function class $\mathcal{F}$ equipped with a distance measure $\mathsf{d}(\cdot,\cdot)$, we define a ζ-covering of $\mathcal{F}$ as a collection of representative functions $f_1,\dots,f_N$ such that, for every $f\in\mathcal{F}$, some f_i satisfies $\mathsf{d}(f,f_i)\le\zeta$. The smallest such N is the ζ-covering number of $\mathcal{F}$ with respect to d. For our analysis, we use the supremum distance $\mathsf{d}(f,g)=\|f-g\|_\infty$.

The critic classes are fixed independently of the replay sample. For $0<\zeta\le H$, let $\mathcal{N}_k(\zeta)$ be a common upper bound on the proper L_∞ covering numbers of $\mathcal{F}_j^u$, for $j\in\{k,k+1\}\cap\{1,\dots,H\}$ and $u\in\{\mathsf{wait},\mathsf{launch}\}$, at radius ζ. Thus, $\mathcal{N}_k$ controls both the current regression class and the classes generating its continuation targets. At $k=H$, the continuation value is identically zero, so no next-stage covering number is needed. Proper covers have representatives in their respective critic classes and hence preserve the clipping bounds. For bounded-parameter ReLU critic classes, standard log covering-number bounds depend on network size, architecture, and the covering radius (Chen et al., 2019; Schmidt-Hieber, 2020).

Distribution shift. We also define a class-restricted distribution shift coefficient. Let ρ_1 and ρ_2 be two visitation measures over tuples (h,x,u). Define the full critic class

$$\mathfrak{F}_k=\{F:\ F(h,x,u)=f^u(h,x),\ f^u\in\mathcal{F}_k^u,\ u\in\{\mathsf{wait},\mathsf{launch}\}\}.$$

At decision stage k, define the class-restricted distribution-shift coefficient

$$C_k(\rho_1,\rho_2)=\sup_{F\in\mathfrak{F}_k,\ V:[0,H]\times\mathcal{X}\to[0,H]}\frac{\mathbb{E}_{(h,x,u)\sim\rho_1}\left[|F(h,x,u)-(\mathcal{T}_k^uV)(h,x)|\right]}{\mathbb{E}_{(h,x,u)\sim\rho_2}\left[|F(h,x,u)-(\mathcal{T}_k^uV)(h,x)|\right]}.$$

Distribution shift is a concept for quantifying the mismatch between the data-sampling distribution and the target distribution under evaluation (Jin et al., 2021; Xie et al., 2021; Cheng et al., 2022). In our case, the training trajectories in Algorithm 1 are obtained by the behavior policy $\pi^{\mathrm{b}}_{\mathrm{meta}}$, which is different from the returned policy $\widehat{\pi}_{\mathrm{meta}}$ and the fixed-generator oracle policy $\pi^{\dagger}_{\mathrm{meta}}$.

Imitation error. Under regularity assumptions, existing literature has established that diffusion models are consistent distribution estimators and attain minimax optimality (Oko et al., 2023; Fu et al., 2024).

C.3 Proof of Theorem 1

Proof. Throughout the proof, $c>0$ denotes an absolute constant that may change from line to line. Let $\pi^\star_{\mathrm{chunk}}$ and $\pi^\star_{\mathrm{meta}}$ be an optimal policy pair attaining $V_1^\star(x_1)$. We decompose the sub-optimality as

$$\mathtt{SubOpt}(\pi_{\mathrm{chunk}},\widehat{\pi}_{\mathrm{meta}})=\mathcal{E}_1+\mathcal{E}_2+\mathcal{E}_3+\mathcal{E}_4,$$

where

$$\begin{aligned}
\mathcal{E}_1&=V_1^\star(x_1)-V_1^{\pi^{\mathrm{exp}}_{\mathrm{chunk}},\pi^\star_{\mathrm{meta}}}(x_1),\\
\mathcal{E}_2&=V_1^{\pi^{\mathrm{exp}}_{\mathrm{chunk}},\pi^\star_{\mathrm{meta}}}(x_1)-V_1^{\pi_{\mathrm{chunk}},\pi^\star_{\mathrm{meta}}}(x_1),\\
\mathcal{E}_3&=V_1^{\pi_{\mathrm{chunk}},\pi^\star_{\mathrm{meta}}}(x_1)-\sup_{\pi_{\mathrm{meta}}}V_1^{\pi_{\mathrm{chunk}},\pi_{\mathrm{meta}}}(x_1)\le 0,\\
\mathcal{E}_4&=\sup_{\pi_{\mathrm{meta}}}V_1^{\pi_{\mathrm{chunk}},\pi_{\mathrm{meta}}}(x_1)-V_1^{\pi_{\mathrm{chunk}},\widehat{\pi}_{\mathrm{meta}}}(x_1).
\end{aligned}$$

In the sequel, we tackle terms $\mathcal{E}_1$ and $\mathcal{E}_2$ similarly. Term $\mathcal{E}_4$ is bounded by empirical process arguments.

• **Bounding $\mathcal{E}_1$ and $\mathcal{E}_2$.** Both terms compare different action generators under a fixed meta control policy. We first prove a generic comparison result for two arbitrary action policies π_a and π'_a, under a fixed meta policy π_{meta}. For notational simplicity, we write

$$V_h^a(x)=V_h^{\pi_a,\pi_{\mathrm{meta}}}(x)\quad\text{and}\quad V_h^{a'}(x)=V_h^{\pi'_a,\pi_{\mathrm{meta}}}(x).$$

We also suppress subscripts and superscripts on π_{meta}, as it is fixed in bounding $\mathcal{E}_1$ and $\mathcal{E}_2$. For any bounded function f, we define

$$\big(\mathcal{T}_h^a f\big)(x) = \sum_{u \in \mathcal{U}_h(x)} \pi_{\mathrm{meta},h}(u \mid x) \mathbb{E}^{\pi_a}\left[r_h(s_h, a_h) + f(x_{h+1}) \,|\, x_h = x, u_h = u\right],$$

where the expectation is over the chunk sampled by π_a if $u = \mathsf{launch}$, as well as the delay and physical transition randomness. Define $\mathcal{T}_h^{a'}$ analogously with π'_a in place of π_a. Then we have

$$V_h^a = \mathcal{T}_h^a V_{h+1}^a \quad \text{and} \quad V_h^{a'} = \mathcal{T}_h^{a'} V_{h+1}^{a'}.$$

We denote the value difference at time h as

$$\Xi_h(x) = V_h^a(x) - V_h^{a'}(x).$$

Substituting these Bellman recursions into the value difference leads to

$$\begin{aligned}\Xi_h(x) &= \big(\mathcal{T}_h^a V_{h+1}^a\big)(x) - \big(\mathcal{T}_h^{a'} V_{h+1}^{a'}\big)(x) \\ &= \big(\mathcal{T}_h^a V_{h+1}^{a'}\big)(x) - \big(\mathcal{T}_h^{a'} V_{h+1}^{a'}\big)(x) \\ &\quad + \int \Xi_{h+1}(x')\, K_h^{\pi_a, \pi_{\mathrm{meta}}}(\mathrm{d}x' \mid x).\end{aligned}$$

For each feasible meta action $u \in \mathcal{U}_h(x)$, we define the one-step value difference as

$$\begin{aligned}\xi_h(x, u) = \mathbb{E}^{\pi_a}&\left[r_h(s_h, a_h) + V_{h+1}^{a'}(x_{h+1}) \,\middle|\, x_h = x, u_h = u\right] \\ &- \mathbb{E}^{\pi'_a}\left[r_h(s_h, a_h) + V_{h+1}^{a'}(x_{h+1}) \,\middle|\, x_h = x, u_h = u\right].\end{aligned}$$

Then by the definition of $\mathcal{T}_h^a$, we have

$$\Xi_h(x) = \sum_{u \in \mathcal{U}_h(x)} \pi_{\mathrm{meta},h}(u \mid x)\big\{\xi_h(x, u) + \mathbb{E}^{\pi_a}\left[\Xi_{h+1}(x_{h+1}) \mid x_h = x, u_h = u\right]\big\}.$$

Iterating this recursion from $h = 1$ to H and using $\Xi_{H+1} = 0$ yields

$$\Xi_1(x_1) = \mathbb{E}^{\pi_a}\left[\sum_{h=1}^{H} \xi_h(x_h, u_h) \,\middle|\, x_1\right] \le \mathbb{E}^{\pi_a}\left[\sum_{h=1}^{H} |\xi_h(x_h, u_h)| \,\middle|\, x_1\right].$$

It remains to bound $|\xi_h(x, u)|$. First consider $u = \mathsf{wait}$, where no new chunk is sampled. Hence the immediate reward $r_h(s_h, a_h)$ and the next state x_{h+1} are identical under different action policies. This leads to

$$\xi_h(x, \mathsf{wait}) = 0.$$

Now consider $u = \mathsf{launch}$. We condition on the same generated action chunk:

$$\begin{aligned}|\xi_h(x, \mathsf{launch})| = \Bigg| &\int \pi_{a,h}(z \mid x) \\ &\quad \times \mathbb{E}\Big[r_h(s_h, a_h) + V_{h+1}^{a'}(x_{h+1}) \,\Big|\, x_h = x, u_h = \mathsf{launch}, z_h = z\Big]\, \mathrm{d}z \\ &- \int \pi'_{a,h}(z \mid x) \\ &\quad \times \mathbb{E}\Big[r_h(s_h, a_h) + V_{h+1}^{a'}(x_{h+1}) \,\Big|\, x_h = x, u_h = \mathsf{launch}, z_h = z\Big]\, \mathrm{d}z\Bigg| \\ \le\ & 2H\, \mathsf{d}_{\mathrm{TV}}\big(\pi_{a,h}(\cdot \mid s_h, B_h), \pi'_{a,h}(\cdot \mid s_h, B_h)\big),\end{aligned}$$

where s_h and B_h are provided in the meta state x. Combining the wait and launch cases, we obtain

$$\begin{aligned}\Xi_1(x_1) &\le 2H \sum_{h=1}^{H} \mathbb{E}^{\pi_a}\left[\mathbb{1}\{u_h = \mathsf{launch}\}\mathsf{d}_{\mathrm{TV}}\left(\pi_{a,h}(\cdot \mid s_h, B_h), \pi'_{a,h}(\cdot \mid s_h, B_h)\right) \mid x_1\right] \\ &\le 2H \sum_{h=1}^{H} \mathbb{E}^{\pi_a}\left[\mathsf{d}_{\mathrm{TV}}\big(\pi_{a,h}(\cdot \mid s_h, B_h), \pi'_{a,h}(\cdot \mid s_h, B_h)\big) \mid x_1\right]. \qquad (1)\end{aligned}$$

We now instantiate the generic bound (1) to $\mathcal{E}_1$ and $\mathcal{E}_2$. Taking

$$\pi_a = \pi^\star_{\mathrm{chunk}}, \quad \pi'_a = \pi^{\mathrm{exp}}_{\mathrm{chunk}}, \quad \pi_{\mathrm{meta}} = \pi^\star_{\mathrm{meta}},$$

we have

$$\mathcal{E}_1 \le 2H \sum_{h=1}^{H} \mathbb{E}^{\pi^\star_{\text{chunk}},\pi^\star_{\text{meta}}} \left[\mathsf{d}_{\text{TV}}\big(\pi^\star_{\text{chunk},h}(\cdot \mid s_h, B_h), \pi^{\text{exp}}_{\text{chunk},h}(\cdot \mid s_h, B_h)\big) \mid x_1 \right]. \tag{2}$$

Similarly, for

$$\pi_a = \pi^{\text{exp}}_{\text{chunk}}, \quad \pi'_a = \pi_{\text{chunk}}, \quad \pi_{\text{meta}} = \pi^\star_{\text{meta}}$$

it holds that

$$\mathcal{E}_2 \le 2H \sum_{h=1}^{H} \mathbb{E}^{\pi^{\text{exp}}_{\text{chunk}},\pi^\star_{\text{meta}}} \left[\mathsf{d}_{\text{TV}}\big(\pi^{\text{exp}}_{\text{chunk},h}(\cdot \mid s_h, B_h), \pi_{\text{chunk},h}(\cdot \mid s_h, B_h)\big) \mid x_1 \right]. \tag{3}$$

By the definitions of $\mathcal{B}_{\text{exp}}$ and $\mathcal{B}_{\text{imit}}$, the two bounds above are controlled by universal constants times $\mathcal{B}_{\text{exp}}$ and $\mathcal{B}_{\text{imit}}$, respectively. These give the first two terms in the upper bound of `SubOpt`.

• **Bounding $\mathcal{E}_4$.** Define the value function $\widehat{V}_k(h_k, x_{h_k}) = \max_{u \in \mathcal{U}_{h_k}(x_{h_k})} \widehat{Q}_k(h_k, x_{h_k}, u)$ and let $\pi^\dagger_{\text{meta}} \in \operatorname{argmax}_{\pi_{\text{meta}}} V_1^{\pi_{\text{chunk}},\pi_{\text{meta}}}(x_1)$ be a fixed-generator oracle meta control policy. Since π_{chunk} is fixed, we omit it in superscripts and subscripts. Then we have

$$\begin{aligned}
\mathcal{E}_4 &= V_1^{\pi^\dagger_{\text{meta}}}(x_1) - V_1^{\widehat{\pi}_{\text{meta}}}(x_1) \\
&= \underbrace{V_1^{\pi^\dagger_{\text{meta}}}(x_1) - \widehat{V}_1(1, x_1)}_{(\clubsuit)} + \underbrace{\widehat{V}_1(1, x_1) - V_1^{\widehat{\pi}_{\text{meta}}}(x_1)}_{(\spadesuit)}
\end{aligned}$$

We bound ($\clubsuit$) and ($\spadesuit$) similarly. At decision stage k, we denote $u^\dagger_{h_k}$ as the optimal meta control action returned by $\pi^\dagger_{\text{meta}}$. It is straightforward that $\widehat{V}_k(h_k, x_{h_k}) \ge \widehat{Q}_k(h_k, x_{h_k}, u^\dagger_{h_k})$. Therefore, we deduce that

$$\begin{aligned}
&V_k^{\pi^\dagger_{\text{meta}}}(h_k, x_{h_k}) - \widehat{V}_k(h_k, x_{h_k}) \\
&\le \mathcal{T}_k V_{k+1}^{\pi^\dagger_{\text{meta}}}(h_k, x_{h_k}, u^\dagger_{h_k}) - \widehat{Q}_k(h_k, x_{h_k}, u^\dagger_{h_k}) \\
&= \mathcal{T}_k V_{k+1}^{\pi^\dagger_{\text{meta}}}(h_k, x_{h_k}, u^\dagger_{h_k}) - \mathcal{T}_k \widehat{V}_{k+1}(h_k, x_{h_k}, u^\dagger_{h_k}) + \mathcal{T}_k \widehat{V}_{k+1}(h_k, x_{h_k}, u^\dagger_{h_k}) - \widehat{Q}_k(h_k, x_{h_k}, u^\dagger_{h_k}) \\
&\le \mathbb{E}\left[V_{k+1}^{\pi^\dagger_{\text{meta}}}(h_{k+1}, x_{h_{k+1}}) - \widehat{V}_{k+1}(h_{k+1}, x_{h_{k+1}}) \mid h_k, x_{h_k}, u^\dagger_{h_k} \right] \\
&\quad + \left| \widehat{Q}_k(h_k, x_{h_k}, u^\dagger_{h_k}) - \mathcal{T}_k \widehat{V}_{k+1}(h_k, x_{h_k}, u^\dagger_{h_k}) \right|,
\end{aligned}$$

where the last inequality cancels the identical reward R_k. Iterating the inequality for $k = 1, \ldots, H$, we have

$$(\clubsuit) \le \sum_{k=1}^{H} \mathbb{E}\left[\left| \widehat{Q}_k(h_k, x_{h_k}, u_{h_k}) - \mathcal{T}_k \widehat{V}_{k+1}(h_k, x_{h_k}, u_{h_k}) \right| \right],$$

where the expectation is taken over the visitation measure $(h_k, x_{h_k}, u_{h_k}) \sim \rho^\dagger_k$ induced by $\pi^\dagger_{\text{meta}}$. For the ($\spadesuit$) term, we denote $\widehat{u}_{h_k}$ as the meta control action returned by $\widehat{\pi}_{\text{meta}}$. Using the fact that $\widehat{V}_k(h_k, x_{h_k}) = \widehat{Q}_k(h_k, x_{h_k}, \widehat{u}_{h_k})$, we have

$$\begin{aligned}
&\widehat{V}_k(h_k, x_{h_k}) - V_k^{\widehat{\pi}_{\text{meta}}}(h_k, x_{h_k}) \\
&= \widehat{Q}_k(h_k, x_{h_k}, \widehat{u}_{h_k}) - \mathcal{T}_k V_{k+1}^{\widehat{\pi}_{\text{meta}}}(h_k, x_{h_k}, \widehat{u}_{h_k}) \\
&= \widehat{Q}_k(h_k, x_{h_k}, \widehat{u}_{h_k}) - \mathcal{T}_k \widehat{V}_{k+1}(h_k, x_{h_k}, \widehat{u}_{h_k}) + \mathcal{T}_k \widehat{V}_{k+1}(h_k, x_{h_k}, \widehat{u}_{h_k}) - \mathcal{T}_k V_{k+1}^{\widehat{\pi}_{\text{meta}}}(h_k, x_{h_k}, \widehat{u}_{h_k}) \\
&\le \left| \widehat{Q}_k(h_k, x_{h_k}, \widehat{u}_{h_k}) - \mathcal{T}_k \widehat{V}_{k+1}(h_k, x_{h_k}, \widehat{u}_{h_k}) \right| \\
&\quad + \mathbb{E}\left[\widehat{V}_{k+1}(h_{k+1}, x_{h_{k+1}}) - V_{k+1}^{\widehat{\pi}_{\text{meta}}}(h_{k+1}, x_{h_{k+1}}) \mid h_k, x_{h_k}, \widehat{u}_{h_k} \right].
\end{aligned}$$

Iterating for $k = 1, \ldots, H$, we obtain

$$(\spadesuit) \le \sum_{k=1}^{H} \mathbb{E}\left[\left| \widehat{Q}_k(h_k, x_{h_k}, u_{h_k}) - \mathcal{T}_k \widehat{V}_{k+1}(h_k, x_{h_k}, u_{h_k}) \right| \right],$$

where the expectation is taken over another visitation measure $(h_k, x_{h_k}, u_{h_k}) \sim \widehat{\rho}_k$ induced by $\widehat{\pi}_{\text{meta}}$. By construction, $\widehat{Q}_k \in \mathfrak{F}_k$, and by clipping, $\widehat{V}_{k+1}$ is bounded in $[0, H]$. Hence the residual

$\widehat{Q}_k - \mathcal{T}_k \widehat{V}_{k+1}$ is covered by the class-restricted coefficient C_k. Invoking the distribution shift coefficient C_k, we have

$$\begin{aligned}\mathcal{E}_4 &\le (\clubsuit) + (\spadesuit)\\ &\le \sum_{k=1}^{H} \big(C_k(\rho_k^\dagger, \rho_k^{\mathrm{b}}) + C_k(\widehat{\rho}_k, \rho_k^{\mathrm{b}})\big)\mathbb{E}_{\rho_k^{\mathrm{b}}}\left[\left|\widehat{Q}_k(h_k, x_{h_k}, u_{h_k}) - \mathcal{T}_k\widehat{V}_{k+1}(h_k, x_{h_k}, u_{h_k})\right|\right]\\ &\le \sum_{k=1}^{H} \big(C_k(\rho_k^\dagger, \rho_k^{\mathrm{b}}) + C_k(\widehat{\rho}_k, \rho_k^{\mathrm{b}})\big)\sqrt{\mathbb{E}_{\rho_k^{\mathrm{b}}}\left[\left|\widehat{Q}_k(h_k, x_{h_k}, u_{h_k}) - \mathcal{T}_k\widehat{V}_{k+1}(h_k, x_{h_k}, u_{h_k})\right|^2\right]}.\end{aligned}$$

It remains to bound the fitted residual $\widehat{Q}_k - \mathcal{T}_k\widehat{V}_{k+1}$. Fix a stage $k \in [H]$ and write

$$I_{k,u} = \{i : K_i \ge k,\ u_{i,k} = u\}, \qquad n_{k,u} = |I_{k,u}|, \qquad u \in \{\mathsf{wait}, \mathsf{launch}\}.$$

Condition only on each trajectory's stage-k selection category: whether it reaches stage k and, if so, which action it takes. This fixes both action counts at this stage. For a fixed action u with $n_{k,u} > 0$, let

$$W_i = (h_{i,k}, x_{i,k}), \qquad W_i' = (h_{i,k+1}, x_{i,k+1}), \qquad i \in I_{k,u}.$$

The selected raw tuples $(W_i, R_{i,k}, W_i')$ are conditionally i.i.d. with the stage-k, action-u replay law: trajectories are i.i.d., and each selection category depends only on its own trajectory. Independence between stages within a trajectory is not needed. Conditional on the current context and action, the SMDP transition follows the Bellman law defining $\mathcal{T}_k^u$. Let P be expectation under this law of a raw tuple (W, R, W'), $P_{n_{k,u}}$ empirical averaging over the selected tuples, and $\mu = \mu_k^u$ the marginal law of W. For data-dependent functions, P integrates a fresh raw tuple while holding the realized functions fixed; it does not denote the conditional law of the in-sample fitted targets.

For $k < H$, consider any $F \in \mathfrak{F}_{k+1}$ and its continuation value

$$V(h, x) = \max_{a \in \mathcal{U}_h(x)} F(h, x, a).$$

At terminal contexts we use zero continuation, and for $k = H$ take $V = 0$. For deterministic $f, g \in \mathcal{F}_k^u$ and a deterministic continuation V of this form, define

$$\begin{aligned}\Delta_{f,g,V}(W, R, W') &= \big(f(W) - R - V(W')\big)^2 - \big(g(W) - R - V(W')\big)^2\\ &= (f - g)(W)\,[f(W) + g(W) - 2R - 2V(W')].\end{aligned}$$

Since $0 \le R \le H$ and the critics and continuations are clipped to $[0, H]$, each squared-loss residual has magnitude at most $2H$. Thus

$$|\Delta_{f,g,V}| \le 4H^2, \qquad \mathrm{Var}_P(\Delta_{f,g,V}) \le P\Delta_{f,g,V}^2 \le 16H^2\|f - g\|_{L^2(\mu)}^2.$$

For each fixed triple, the centered summands are conditionally independent, mean-zero, and bounded in magnitude by $8H^2$. Bernstein's inequality followed by Young's inequality therefore gives, with failure probability at most $2e^{-t}$,

$$\big|(P_{n_{k,u}} - P)\Delta_{f,g,V}\big| \le \frac{1}{32}\|f - g\|_{L^2(\mu)}^2 + cH^2\frac{t}{n_{k,u}}.$$

Fix $0 < \zeta \le H$. Choose proper covers $\mathcal{C}_{j,a}(\zeta)$ deterministically from the fixed critic classes, each of cardinality at most $\mathcal{N}_k(\zeta)$, for $j \in \{k, k+1\} \cap \{1, \dots, H\}$ and $a \in \{\mathsf{wait}, \mathsf{launch}\}$. Pointwise maximization over the fixed feasible-action sets is 1-Lipschitz in the supremum norm. A pair of next-stage cover representatives therefore approximates every continuation above within ζ. Two current-stage representatives for f, g and the two next-stage representatives form a family of size at most $\mathcal{N}_k(\zeta)^4$. At $k = H$, only the two current choices are needed, and the same upper bound applies since $\mathcal{N}_k(\zeta) \ge 1$. Taking

$$t = 4\log\mathcal{N}_k(\zeta) + \log(2/\delta_{k,u})$$

and a union bound gives a single event with conditional failure probability at most $\delta_{k,u}$ for this finite family. Uniformity in g is needed because the Bellman-completeness comparator will depend on the learned continuation.

We extend this event to all $f, g \in \mathcal{F}_k^u$ and all continuations induced by $\mathfrak{F}_{k+1}$ when $k < H$, or $V = 0$ when $k = H$. Replacing each prediction and continuation by its cover representative changes the loss difference pointwise by at most $cH\zeta$: each residual is bounded by $2H$, and its prediction and continuation each change by at most ζ. The squared $L^2(\mu)$ distance between the two current

representatives is at most $2\|f-g\|^2_{L^2(\mu)} + 8\zeta^2$. Using $\zeta^2 \le H\zeta$ and adjusting universal constants yields

$$\begin{aligned}\left|(P_{n_{k,u}} - P)\Delta_{f,g,V}\right| \le \frac{1}{8}\|f-g\|^2_{L^2(\mu)} \\ + c\left[H^2\frac{\log \mathcal{N}_k(\zeta) + \log(2/\delta_{k,u})}{n_{k,u}} + H\zeta\right].\end{aligned} \tag{4}$$

The factor four in the entropy is absorbed into c. This event is uniform over the full current function classes and all such continuations, with $V = 0$ at $k = H$.

On this event, set

$$\widehat{f} = \widehat{Q}^u_k, \qquad m = \mathcal{T}^u_k \widehat{V}_{k+1}.$$

By the clipping convention, $\widehat{V}_{k+1}$ satisfies the range condition in Assumption 2. Apply Bellman completeness to the realized $\widehat{V}_{k+1}$ to choose $f^\circ \in \mathcal{F}^u_k$ with $\|f^\circ - m\|_\infty \le \epsilon$. The uniform event permits substituting $(f, g, V) = (\widehat{f}, f^\circ, \widehat{V}_{k+1})$, including the data-dependent comparator. Empirical risk minimization gives

$$P_{n_{k,u}}\Delta_{\widehat{f},f^\circ,\widehat{V}_{k+1}} \le 0. \tag{5}$$

For fixed functions, fresh-tuple expectation gives the functional identity

$$P\Delta_{f,g,V} = \|f - \mathcal{T}^u_k V\|^2_{L^2(\mu)} - \|g - \mathcal{T}^u_k V\|^2_{L^2(\mu)}. \tag{6}$$

This identity also holds for the realized learned functions. Combining it with (5) and (4), we obtain

$$\begin{aligned}\|\widehat{f} - m\|^2_{L^2(\mu)} - \|f^\circ - m\|^2_{L^2(\mu)} &\le \frac{1}{8}\|\widehat{f} - f^\circ\|^2_{L^2(\mu)} \\ &\quad + c\left[H^2\frac{\log \mathcal{N}_k(\zeta) + \log(2/\delta_{k,u})}{n_{k,u}} + H\zeta\right] \\ &\le \frac{1}{4}\|\widehat{f} - m\|^2_{L^2(\mu)} + \frac{1}{4}\|f^\circ - m\|^2_{L^2(\mu)} \\ &\quad + c\left[H^2\frac{\log \mathcal{N}_k(\zeta) + \log(2/\delta_{k,u})}{n_{k,u}} + H\zeta\right].\end{aligned}$$

Absorbing the first squared error and using $\|f^\circ - m\|_{L^2(\mu)} \le \epsilon$ gives

$$\|\widehat{Q}^u_k - \mathcal{T}^u_k \widehat{V}_{k+1}\|_{L^2(\mu^u_k)} \le c\epsilon + c\left[H^2\frac{\log \mathcal{N}_k(\zeta) + \log(2/\delta_{k,u})}{n_{k,u}} + H\zeta\right]^{1/2}. \tag{7}$$

Set $\delta_{k,u} = \delta/(2H)$. Conditional on the stage-k selection pattern with both action counts positive, a union bound over the two actions has failure probability at most δ/H. No independence between these two events is required. The replay law ρ^{b}_k decomposes into the conditional state-time laws μ^u_k, so

$$\|\widehat{Q}_k - \mathcal{T}_k \widehat{V}_{k+1}\|^2_{L^2(\rho^{\mathrm{b}}_k)} \le \max_{u \in \{\mathsf{wait},\mathsf{launch}\}} \|\widehat{Q}^u_k - \mathcal{T}^u_k \widehat{V}_{k+1}\|^2_{L^2(\mu^u_k)}.$$

Consequently, on the same conditional event,

$$\|\widehat{Q}_k - \mathcal{T}_k \widehat{V}_{k+1}\|_{L^2(\rho^{\mathrm{b}}_k)} \le c\epsilon + c\left(H^2\alpha_k(\zeta,\delta) + H\zeta\right)^{1/2}, \tag{8}$$

where

$$\alpha_k(\zeta,\delta) = \max_{u \in \{\mathsf{wait},\mathsf{launch}\}} \frac{\log \mathcal{N}_k(\zeta) + \log(4H/\delta)}{n_{k,u}}.$$

Conditional on the stage-k selection pattern, both action counts are fixed. We may therefore choose a deterministic radius arbitrarily close to the infimum of $H^2\alpha_k(\zeta,\delta) + H\zeta$ over $0 < \zeta \le H$. For each positive optimization tolerance, applying the two-action bound at such a radius bounds the same fitted residual by a threshold depending only on this infimum and the tolerance. Letting the tolerance decrease to zero preserves the probability bound, by continuity for the resulting decreasing residual-threshold events. Averaging over the selection pattern and taking a union bound over k gives, with probability at least $1-\delta$, the optimized residual bound simultaneously at every stage with positive action counts. This argument optimizes the radius separately at each stage and requires no independence across stages.

Substituting these optimized residual bounds into the preceding distribution-shift decomposition gives, whenever all action counts are positive,

$$\mathcal{E}_4 \le c \sum_{k=1}^{H} \kappa_k \left[\epsilon + \inf_{0<\zeta\le H} \left(H^2 \alpha_k(\zeta, \delta) + H\zeta \right)^{1/2} \right], \tag{9}$$

where

$$\kappa_k = C_k(\rho_k^{\dagger}, \rho_k^{\mathrm{b}}) + C_k(\widehat{\rho}_k, \rho_k^{\mathrm{b}}).$$

Combining (9) with the bounds on $\mathcal{E}_1, \mathcal{E}_2$ and $\mathcal{E}_3 \le 0$ completes the proof. □

D Experimental Details

Unless otherwise stated, all experiments were run on NVIDIA RTX A6000 GPUs.

D.1 Hidden-Danger GridWorld

Environment. The Hidden-Danger GridWorld is implemented with Gymnasium (Towers et al., 2024). The primitive action space is $\{\mathrm{up}, \mathrm{down}, \mathrm{left}, \mathrm{right}\}$. We use two grid sizes: 16×16, with start $(1, 14)$ and goal $(14, 1)$, and 32×32, with start $(1, 30)$ and goal $(30, 1)$. Each grid contains a fixed visible zig-zag road from start to goal, which gives the action generator a default route while allowing off-road detours. The local observation contains a $7 \times 7 \times 6$ patch (boundary, empty cell, agent, revealed danger, visible goal, and nominal road), a normalized goal vector, and normalized time.

Each danger zone is a 3×3 lethal core. A danger zone is hidden until the agent is within L1 distance 3 of the core; after reveal, it remains visible. Danger centers are sampled from middle road cells, excluding the first and last 4 road cells on 16×16 maps and the first and last 6 road cells on 32×32 maps, with small offsets $(0, 0), (\pm 1, 0), (0, \pm 1)$. The main evaluation uses danger counts $\{1, 2, 3, 4\}$ and 512 fixed maps for each grid size and count, with a minimum danger-center distance of 4.

Fixed action generator. The action generator π_{chunk} is a scripted local planner using the current observation, visible road, and revealed danger cells. It treats unrevealed danger cells as empty. At each launch, π_{chunk} outputs a primitive action chunk of length $L = 8$. The first $p = 2$ entries copy the currently committed protected prefix. The planner then simulates this prefix to the handoff state and plans the remaining $L - p$ actions from that state.

Planning uses a road-following shortest-path routine. Revealed danger cells are blocked. Among feasible cells, the planner uses step cost

$$1 + \lambda_{\mathrm{off}} \mathbf{1}\{\text{cell is off the nominal road}\}, \qquad \lambda_{\mathrm{off}} = 0.25.$$

Equivalently, the visible road is a cheap default route. When no danger is revealed, π_{chunk} follows the road. When a revealed danger blocks the road, the same planner takes a short off-road detour around the revealed core and then rejoins the next reachable road cell.

Reward and delayed execution. The reward is

$$r_t = 0.05\big(d_{\mathrm{L1}}(s_t, \mathrm{goal}) - d_{\mathrm{L1}}(s_{t+1}, \mathrm{goal})\big) + \mathbf{1}\{s_{t+1} = \mathrm{goal}\} - \mathbf{1}\{s_{t+1} \in \mathrm{danger}\}.$$

Progress toward the goal contributes positive reward. Entering a danger core incurs a -1 terminal penalty; reaching the goal gives a $+1$ terminal bonus. We use $\Delta_h(s_h) \equiv 2$, protection length $p = 2$, and $L = 8$. The episode horizon is 64 for 16×16 maps and 128 for 32×32 maps.

Meta-control network and training. The meta action space is binary: wait or launch. The Q network is a two-hidden-layer MLP with width 256 and ReLU activations, followed by two outputs $Q(x, \mathsf{wait})$ and $Q(x, \mathsf{launch})$. We train π_{meta} online with Algorithm 1 using a one-step SMDP TD target. Behavior policy rollouts use a mixture policy: 50% relaunch-on-reveal, 30% balanced free decisions that choose wait or launch with equal probability when both are feasible, and 20% launch whenever feasible. At evaluation time, the learned policy greedily chooses the feasible meta action with larger Q value. Active launch rate excludes the initial seeded chunk and therefore measures voluntary launches during the episode. We evaluate one frozen checkpoint per grid size, each trained for 3000 episodes.

Quantitative evaluation. For each grid size and danger count, we compare learned timing with the Local Planner's default launch schedule on the same 512 maps, keeping the planner fixed. This extends the rollout in Section 4.2 to 4,096 maps with varying amounts of hidden danger.

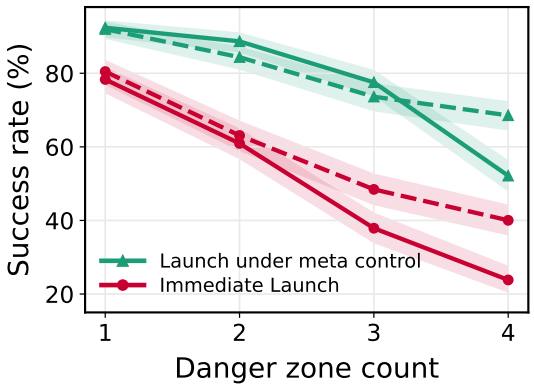

Figure 3: Success rate in hidden-danger GridWorld. Solid lines denote 16×16 grids and dashed lines denote 32×32 grids; shaded regions are Wilson 95% confidence intervals.

Figure 3 and Table 5 show that learned launch timing improves success in all eight configurations, with absolute SR gains of 11.52–39.65 percentage points. Across the equally sized configurations, success increases from 54.1% to 78.7%. The same planner therefore reaches the goal more often when the meta policy chooses when to request a new route.

Table 5: Hidden-Danger success on the same 512 maps per configuration. Brackets give Wilson 95% confidence intervals for each policy; gains are absolute differences in SR (percentage points).

Grid	Dangers	Local Planner SR (%)	Local Planner w/ π_{meta} SR (%)	ΔSR (pp)
16×16	1	78.32 [74.55, 81.67]	92.38 [89.76, 94.38]	+14.06
	2	60.94 [56.64, 65.07]	88.67 [85.63, 91.13]	+27.73
	3	37.89 [33.79, 42.17]	77.54 [73.73, 80.94]	+39.65
	4	23.83 [20.34, 27.70]	52.15 [47.82, 56.44]	+28.32
32×32	1	80.47 [76.81, 83.67]	91.99 [89.32, 94.04]	+11.52
	2	63.09 [58.82, 67.15]	84.38 [80.98, 87.26]	+21.29
	3	48.44 [44.14, 52.76]	73.63 [69.65, 77.26]	+25.20
	4	40.04 [35.88, 44.34]	68.55 [64.41, 72.43]	+28.52

Active launch rate. Table 6 reports the active launch rate under the learned meta control policy π_{meta}. We average each episode's active launches per physical environment step, excluding the initial seeded chunk. Under the Local Planner's default launch rule, this rate is 0.5 because we set $\Delta = 2$, so the controller launches once every two steps whenever inference is free. Across all 4096 episodes, the mean active launch rate decreases from 0.50 to 0.30, a 40.3% reduction; per-configuration reductions range from 29.2% to 47.5%. Together with the improved success rates in Figure 3, these results show that the learned meta controller improves performance while launching substantially less often.

Table 6: Active launch rate under meta control (↓). Entries are mean ± across-episode standard deviation; parentheses show reduction from the Local Planner.

Danger count	16×16	32×32
1	0.29 ± 0.04 (−41.5%)	0.26 ± 0.02 (−47.5%)
2	0.31 ± 0.04 (−38.0%)	0.27 ± 0.02 (−46.4%)
3	0.34 ± 0.03 (−32.4%)	0.28 ± 0.03 (−44.6%)
4	0.35 ± 0.04 (−29.2%)	0.29 ± 0.03 (−42.8%)

Reveal-aligned launch behavior. We align the learned policy's decisions to the first free decision after the first danger reveal, when no inference is pending and both meta actions are feasible. Across the same 4096 maps, the policy chooses wait at 81.6% of pre-reveal free decisions (95% CI: 81.3–82.0%) and launch at the first free post-reveal decision in 96.4% of episodes (95% CI: 95.8–96.9%). The mean Q-gap, $Q(x, \mathsf{launch}) - Q(x, \mathsf{wait})$, changes from -1.10×10^{-2} before reveal to 1.49 at that first free decision. Table 7 shows the corresponding decision-relative profile: the launch preference rises sharply when danger information becomes available and then subsides. Together with the higher success rate, this pattern shows that the learned policy uses newly revealed danger to time requests for avoidance actions.

Table 7: Behavior around the first free post-reveal decision (offset 0). Offsets count free decisions. Each row reports its contributing maps; Q-gaps and their intervals are scaled by 100; intervals use 5000 map-bootstrap resamples.

Decision offset	Maps	$100\times$ mean Q-gap [95% CI]	Launch rate (%)
-3	2969	-1.50 $[-1.60, -1.50]$	15.9
-2	3478	-1.40 $[-1.40, -1.30]$	9.7
-1	4096	-1.30 $[-1.40, -1.30]$	20.6
0	4096	149.00 [144.20, 153.80]	96.4
1	3961	5.30 [4.60, 6.10]	78.9
2	3888	-4.60 $[-5.40, -3.80]$	26.1

D.2 Launch-Timing Baselines

All timing policies use the same frozen action generator, buffer-update rule, and feasible action set $\mathcal{U}_h(x_h)$. While inference is pending, buffered execution continues without a new launch. At a no-pending state with only one feasible action, every policy takes that action. The rules below specify the choice when both wait and launch are feasible. A startup prefill initializes the buffer and counts as one generator call.

Default timing and Wait-Forced. We label the unmodified baselines by their action generators: TrainingRTC, FutureRTC, and DynamicVLA. Each chooses launch when both actions are feasible; Wait-Forced always chooses wait. Thus Wait-Forced launches only when the feasible action set requires it.

Slack-Uniform. At the start of each free decision window, let n_{safe} be the number of feasible launch offsets, including the current tick. The largest allowable voluntary waiting duration is $n_{\text{safe}} - 1$. Slack-Uniform draws once,

$$W \sim \mathrm{Unif}\{0, \dots, n_{\text{safe}} - 1\},$$

waits for W voluntary decision steps, and then launches. Each voluntary Wait decrements the remaining count by one; any launch clears it. The next free window receives a new draw. The shared feasible action set takes precedence throughout.

Periodic. Let h_0 be the first no-pending decision tick after prefill. For an integer period $K \in \{1, \dots, 8\}$, Periodic chooses

$$u_h = \begin{cases} \mathsf{launch}, & (h - h_0) \bmod K = 0, \\ \mathsf{wait}, & \text{otherwise.} \end{cases}$$

The clock runs in physical time and is not reset by arrivals or forced launches. Scheduled ticks that occur while inference is pending are skipped.

State-Delta (RTC). Let τ denote the most recent actual launch, initialized by prefill. We estimate a standard deviation σ_j for each physical-state coordinate from a TrainingRTC calibration rollout bank. Let J contain the coordinates that vary in this bank and set $d_j = \max\{\sigma_j, 10^{-3}\}$. State-Delta launches when the normalized RMS state change exceeds a threshold θ:

$$D_h^{\text{state}} = \left[\frac{1}{|J|} \sum_{j \in J} \left(\frac{s_{h,j} - s_{\tau,j}}{d_j} \right)^2 \right]^{1/2}, \qquad u_h = \begin{cases} \mathsf{launch}, & D_h^{\text{state}} > \theta, \\ \mathsf{wait}, & \text{otherwise.} \end{cases}$$

The reference updates on every actual launch, including a forced launch, and stays fixed on Wait and on arrival.

RTC calibration. For each task–delay pair, 2,048 TrainingRTC episodes determine the state normalization and the State-Delta threshold grid $\{0, Q_{.25}, Q_{.50}, Q_{.75}, Q_{.90}, 10^6\}$, with duplicate values removed. The quantiles use state-change scores at decisions where both actions are feasible. For each of the three meta-policy seeds, we evaluate every candidate period or threshold on 2,048 separate calibration episodes. Let $\bar{C}_\theta$ and $\bar{S}_\theta$ denote its mean calls and success, and let $\bar{C}_{\text{meta}}$ be the corresponding meta-policy call target. Among candidates satisfying

$$|\bar{C}_\theta - \bar{C}_{\text{meta}}| \le 0.05\bar{C}_{\text{meta}},$$

we choose the candidate with the highest success rate, breaking ties by fewer calls and then the smaller parameter. If none meets the band, we choose the closest call count, then higher success, then the smaller parameter. Periods with identical calibration decision/action traces share their smallest representative. These choices are frozen before the final evaluation.

Visual-Delta (DOM). Let $f_h^w, f_h^o \in \mathbb{R}^{960}$ be the mean-pooled pre-fusion features from the wrist and overhead cameras, and let r_w, r_o be their respective feature-block RMS values saved during meta-policy training. Define

$$v_h = \left[\frac{f_h^w}{r_w}; \frac{f_h^o}{r_o}\right] \in \mathbb{R}^{1920}, \qquad D_h^{\text{visual}} = \frac{\|v_h - v_\tau\|_2}{\sqrt{1920}}.$$

Visual-Delta chooses launch if $D_h^{\text{visual}} > \theta$ and wait otherwise. As for State-Delta, τ indexes the most recent actual launch, including prefill and forced launches. A launch reuses the acquired visual features; waiting does not change the reference.

We calibrate one threshold per task and model seed on $\text{Uniform}_{4:8}$, using four training scenes and five trials per scene. The candidate grid is $\{0, Q_{.25}, Q_{.50}, Q_{.75}, Q_{.90}, +\infty\}$, where the score distribution weights episodes equally and decisions equally within each episode. The selected threshold minimizes $|\bar{C}_\theta - \bar{C}_{\text{meta}}|$, with exact ties favoring the larger threshold. Calibration uses calls including prefill. The thresholds for seeds 0 and 1 are $(0, 107008) \times 10^{-6}$ on CR and $(111478, 102092) \times 10^{-6}$ on DA; they transfer unchanged to Point_6 and the empirical delay law.

D.3 Kinetix Evaluation and Full Comparisons

Action generator. We train the Training-Time RTC backbone (Black et al., 2025b) with a VLaSH-inspired recipe (Tang et al., 2025): each training example supervises multiple branches with different execution offsets, jointly learning the action suffixes conditioned on their corresponding committed prefixes.

Meta-policy network and training. The RTC meta network encodes physical state, buffered actions, and timing scalars with tanh MLPs of widths $(256, 128)$, $(128, 128)$, and $(64, 64)$, respectively. It averages valid action-token embeddings and combines the three encodings in a $(256, 256)$ MLP, followed by success and dense-reward Q heads, each predicting values for Wait and Launch. We fit the network with Algorithm 1 on an offline dataset collected using TrainingRTC, a fixed waiting schedule, balanced Wait/Launch exploration, and Slack-Uniform. These behaviors cover different launch times and buffer states, providing experience for both meta actions. Each RTC policy uses approximately 28,000 physical transitions and greedily selects the feasible action with the highest combined score from its two Q heads. Selection and final testing use separate rollout seeds generated from distinct root seeds.

Delays and repetitions. The fixed-delay laws are Point_d for $d \in \{0, 1, 2, 3\}$. Over this ordered support, Uniform, Geo, RevGeo, and Binom have probabilities $(1, 1, 1, 1)/4$, $(8, 4, 2, 1)/15$, $(1, 2, 4, 8)/15$, and $(1, 3, 3, 1)/8$, respectively. The executor uses protection $p = d$ for Point_d and $p = 3$ for stochastic delays.

Each RTC task–delay pair uses three independently trained meta policies, with 2,048 evaluation episodes per method and seed. This gives $12 \times 3 \times 2{,}048 = 73{,}728$ episodes per method for each delay law, and 589,824 across all eight laws. Heuristics use the corresponding three evaluation seeds and frozen calibration choices. Means weight task–delay–seed combinations equally. Q1 and Q2

each contain 24 task–delay pairs, ranked by independent TrainingRTC evaluations, and 147,456 episodes per method.

FutureRTC uses separate three-member meta-policy ensembles for fixed and random delays. The fixed-delay panel contains one ensemble per task–delay pair and 2,048 episodes per method and pair: 98,304 episodes per method over 48 pairs. The random-delay panel contains three independently trained ensembles per pair, each evaluated for 2,048 episodes: 294,912 episodes per method over 48 pairs. Each three-member ensemble acts as one controller. For Table 2, we first average evaluation replicates within each pair and then weight all 96 pairs equally. Difficulty quartiles are defined independently within the two panels; the combined Q1 and Q2 each contain 12 fixed-delay and 12 random-delay pairs.

Full timing comparison. Tables 8 and 9 retain all timing baselines. Meta control has the highest mean success in seven of eight delay laws and nearly matches State-Delta on RevGeo. Across laws, its mean SR is 87.17%, compared with 85.70% for Periodic and 86.36% for State-Delta at similar mean Calls. Wait-Forced and Slack-Uniform make fewer calls, with lower success.

Table 8: Kinetix SR (%, ↑). All methods use the same TrainingRTC backbone. Green denotes meta control and red denotes TrainingRTC; bold marks the highest mean in each column.

Method / Delay	Point_0	Point_1	Point_2	Point_3	Uniform	Geo	RevGeo	Binom	Avg.
TrainingRTC w/ π_{meta}	**92.74**	**89.75**	**87.15**	**84.72**	**85.84**	**86.54**	84.88	**85.78**	**87.17**
TrainingRTC	90.90	88.53	85.90	83.78	84.85	85.59	84.35	85.26	86.15
Wait-Forced	88.64	85.81	84.19	81.69	81.73	81.59	81.56	81.61	83.35
Slack-Uniform	89.85	86.98	85.00	82.80	83.29	83.57	82.92	83.27	84.71
Periodic	91.11	87.88	85.61	83.43	84.50	84.85	83.96	84.27	85.70
State-Delta	90.64	88.05	86.45	84.30	85.44	85.79	**84.89**	85.32	86.36

Table 9: Mean action-model calls (↓) per episode on Kinetix. Each entry uses the same episodes as Table 8.

Method / Delay	Point_0	Point_1	Point_2	Point_3	Uniform	Geo	RevGeo	Binom	Avg.
TrainingRTC w/ π_{meta}	60.40	57.57	38.97	29.73	45.09	55.30	35.61	45.53	46.02
TrainingRTC	87.42	95.21	50.25	34.90	58.26	79.16	44.29	62.46	63.99
Wait-Forced	19.11	20.02	20.50	20.84	20.90	20.94	20.90	20.93	20.52
Slack-Uniform	30.81	32.77	29.07	26.14	30.72	32.93	28.44	31.25	30.26
Periodic	61.12	55.61	39.27	29.92	46.72	55.03	35.67	45.24	46.07
State-Delta	61.18	58.32	38.87	29.73	45.17	55.21	35.76	45.64	46.23

D.4 Kinetix Network-Input Ablation

We ablate the physical-state and action-buffer inputs to the meta-control Q network. *No state* zeros the physical-state vector before encoding; *no buffer* zeros both action-token values and their validity indicators. Timing scalars remain available in both variants. The masks apply to the meta network during training and evaluation; the RTC generator and executor retain their original observations and buffers.

Protocol. For each task–delay–seed combination, the full-input reference (Full-matched) and both ablations are trained from the same initialization on a shared replay dataset, with the same network size and training budget. Each variant learns its own Bellman targets over 20 fitted iterations and selects its checkpoint using 512 evaluation episodes per candidate. Full-matched is the reference trained for this ablation study, separately from the main-text policy bank.

Evaluation covers 12 tasks, eight delay laws, and three independent training seeds, with 2,048 episodes per variant and task–delay–seed combination: 589,824 episodes per variant. Q1 contains the 24 hardest task–delay pairs. Means weight all task–delay–seed combinations equally.

Table 10: Kinetix input ablations with a fixed RTC action generator. Success is a percentage; calls are mean action-model invocations per episode. The full-input row is the Full-matched reference. Q1 ΔSR is the absolute SR difference (percentage points) from this reference in the hardest quartile.

Meta-network inputs	SR ↑	Calls ↓	Q1 ΔSR ↑
TrainingRTC w/ π_{meta} (full)	**86.78**	43.14	0.00
TrainingRTC w/ π_{meta} (no state)	86.36	43.97	−1.23
TrainingRTC w/ π_{meta} (no buffer)	86.50	45.43	−1.20

Results. Using both state and buffer inputs yields the highest mean SR, 86.78%, and the fewest action-model calls, 43.14 per episode, among the three variants (Table 10). Removing physical state reduces mean SR to 86.36% and increases Calls by 0.83 per episode; removing buffer inputs reduces SR to 86.50% and increases Calls by 2.29. On the hardest quartile, the absolute SR decreases are larger: 1.23 percentage points without state and 1.20 percentage points without buffer. State removal lowers mean success in all four quartiles, whereas the buffer loss is concentrated in the hardest one. These results support conditioning launch timing on both the current physical state and the buffered actions that will execute while inference is deferred, with the clearest benefit in difficult settings.

D.5 DOM Evaluation and Full Comparisons

Meta-policy network and training. The meta network encodes the frozen VLA's pooled pre-fusion features, ordered action buffer, and timing scalars with MLPs of widths $(256, 128)$, $(128, 128)$, and $(64, 64)$. A $(256, 256)$ MLP combines them and outputs $Q(x, \mathsf{wait})$ and $Q(x, \mathsf{launch})$. Hidden layers use layer normalization and ReLU. As in Kinetix, we fit the policy on an offline dataset collected with several timing behaviors to cover different decision states: DOM collection begins with DynamicVLA and fixed-wait episodes, then uses Slack-Uniform. Each DOM policy uses approximately 28,000 physical transitions and greedily selects the feasible action with the higher Q value. Model selection and final testing use separate rollout seeds generated from distinct root seeds.

The two camera views are encoded sequentially. Their prefix features support the timing decision and are reused on Launch, which additionally runs KV fill and flow generation. This is the serial-lazy execution path profiled in Table 4.

Delays and repetitions. Point_6 always takes six ticks; $\text{Uniform}_{4:8}$ samples each of five delays equally. The empirical law assigns probabilities $(1037, 141, 19, 2, 1)/1200$ to delays $(6, 7, 8, 9, 10)$. It is obtained from 1,200 profiled action-generation latencies, each rounded up to a multiple of 40 ms, and has a mean of 6.16 ticks. Evaluation samples delays from these laws at a control frequency of 25 Hz.

For each task–delay–method combination, we evaluate two independently trained meta-policy seeds on ten official scenes, with 20 trials per scene and seed. Each entry therefore contains $10 \times 2 \times 20 = 400$ episodes; CR and DA over three delay laws give 2,400 episodes per method. Baselines use the same scene, seed, and trial identities. Visual-Delta calibration uses 20 episodes per task–seed–candidate, totaling 480 episodes for six candidates, two tasks, and two seeds.

Full timing comparison. Tables 11 and 12 include the Visual-Delta baseline. On CR, this simple feature-change trigger is highly competitive: its SR exceeds that of learned timing by 6.44% under uniform delays and 8.00% under fixed delays, while their empirical-delay SRs are close (49.25% versus 50.25%). On DA, learned timing instead improves SR over Visual-Delta by 17.69–82.95% across the three delay laws. These relative comparisons distinguish ordinary tracking, where a feature-change trigger already works well, from abrupt adaptation, where learned timing provides a larger benefit. Both controllers reduce Calls relative to DynamicVLA.

Table 11: DOM SR (%, ↑). Rows are grouped by task; bold marks the highest success for each task and delay.

Task	Method	$\text{Uniform}_{4:8}$	Point_6	Empirical
CR	DynamicVLA w/ π_{meta}	50.50	50.00	**50.25**
CR	DynamicVLA	40.75	43.50	44.00
CR	Visual-Delta	**53.75**	**54.00**	49.25
DA	DynamicVLA w/ π_{meta}	**42.00**	**38.25**	**40.25**
DA	DynamicVLA	28.50	24.25	25.00
DA	Visual-Delta	33.50	32.50	22.00

Table 12: Mean prefix evaluations / full action-model calls (↓) per DOM episode, including startup prefill. Columns follow Table 11.

Task	Method	$\text{Uniform}_{4:8}$	Point_6	Empirical
CR	DynamicVLA w/ π_{meta}	95.20 / 25.30	91.66 / 24.34	86.59 / 23.99
CR	DynamicVLA	37.76 / 37.76	35.87 / 35.87	34.28 / 34.28
CR	Visual-Delta	50.00 / 31.08	51.24 / 31.40	50.18 / 31.19
DA	DynamicVLA w/ π_{meta}	71.44 / 33.35	70.62 / 33.21	69.82 / 33.62
DA	DynamicVLA	40.05 / 40.05	40.17 / 40.17	40.67 / 40.67
DA	Visual-Delta	78.24 / 32.53	75.41 / 32.92	78.19 / 34.15

Observation and generation costs. Table 12 separates observation-prefix evaluations from complete action-model calls. Meta control checks observations more often: on CR it uses 86.59–95.20 prefix evaluations and 23.99–25.30 action-model calls per episode, compared with 34.28–37.76 of each for DynamicVLA. On DA, the corresponding counts are 69.82–71.44 and 33.21–33.62, compared with 40.05–40.67 of each for DynamicVLA. Visual-Delta also spends additional prefix evaluations on deciding when to launch.

These additional checks use the inexpensive observation and decision path in Table 4; the full action-model calls also incur KV fill and flow generation. The complete Wait path takes 18.76 ms, compared with 234.12 ms for a Launch; the meta decision itself takes only 0.88 ms. This lets the controller check new information repeatedly at a small fraction of the cost of generating another action chunk.

Latency repetitions. The serial-lazy measurements in Table 4 use 16 real inputs, eight each from CR and DA, on an RTX A6000. Each whole path is repeated 50 times per input (800 observations per path); each component is repeated five times (80 observations per component). We report host wall-clock median and 95th-percentile latency from prepared CPU input to a CPU-ready decision or chunk. Subtotal and total are measured directly; component quantiles are not added together.