MIND THE EXECUTION GAP: ACTION-SEMANTIC MISMATCH IN WORLD-MODEL CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

World-model controllers rely on action-conditioned dynamics for prediction and planning, yet real control systems often execute commands asynchronously due to communication delay, packet loss, reordering, and actuator buffering. We study how asynchronous execution changes the action semantics assumed within world-model controllers, rather than treating it only as an external control disturbance. Through controlled interventions, we identify two architecture-dependent failure modes: planning-based controllers such as TD-MPC2 suffer from a future-action timeline mismatch between imagined and executed action sequences, while recurrent world models such as DreamerV3 can attribute observed transitions to commands that were not actually applied. Our analysis shows that TD-MPC2 requires the correct future action sequence during latent dynamics rollout, whereas DreamerV3 requires timely attribution of each transition to the action that generated it. Based on these findings, we introduce two lightweight execution-consistent interfaces, Future-Sequence for TD-MPC2 and Applied-Action Feedback for DreamerV3, that correct these mismatches without modifying the pretrained world models. Experiments across delays, packet loss, reordering, multiple control domains, measured network traces, and a process-separated asynchronous stack consistently support both diagnoses and the corresponding architecture-specific corrections.

1 INTRODUCTION

World models have become a central paradigm for model-based decision making, enabling agents to predict future outcomes, optimize policies through imagined trajectories, and plan actions online in learned latent dynamics (Hansen et al., 2022; Hafner et al., 2024; Hansen et al., 2024; Li et al., 2026; Anand et al., 2026). These capabilities are particularly attractive for continuous control and robotics, where the controller must repeatedly predict how actions change the environment before acting (Wu et al., 2023; Brito & McNamee, 2025; Li et al., 2025). However, practical control systems are rarely perfectly synchronous: action computation, communication, buffering, and actuation may occur on different timelines (Zhang et al., 2013; Park et al., 2018; Wang et al., 2026). As a result, the action issued by a controller is not always the action that actually drives the corresponding environment transition (Katsikopoulos & Engelbrecht, 2003; Bouteiller et al., 2021; Chen et al., 2021).

Prior work on delayed and asynchronous control has followed several directions at different stages of the control loop. Early formulations recover a Markovian decision process by augmenting the state with delayed observations, pending actions, or action histories (Katsikopoulos & Engelbrecht, 2003; Walsh et al., 2009; Derman et al., 2023). Later work uses learned dynamics or belief estimation to compensate for delayed interaction by propagating the effects of pending actions, reconstructing the current latent state, or incorporating multi-step action delay into model-based planning (Chen et al., 2021; Kim et al., 2023; Valensi et al., 2024). More recent methods address stochastic or unobservable execution timing through delay-corrected learning, adaptive interaction models, or alternative future actions conditioned on uncertain execution times and packet loss (Bouteiller et al., 2021; Wu et al., 2024; Wikman et al., 2026). In parallel, recent robot-control systems consider the effects of slow policy inference on execution: RTC overlaps action generation with execution while preserving committed actions, REMAC addresses inconsistencies between current perception and concurrently executed action chunks, and FutureRTC predicts the state or observation available when new actions begin execution (Black et al., 2025; Wang et al., 2026; Jiang et al., 2026).

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

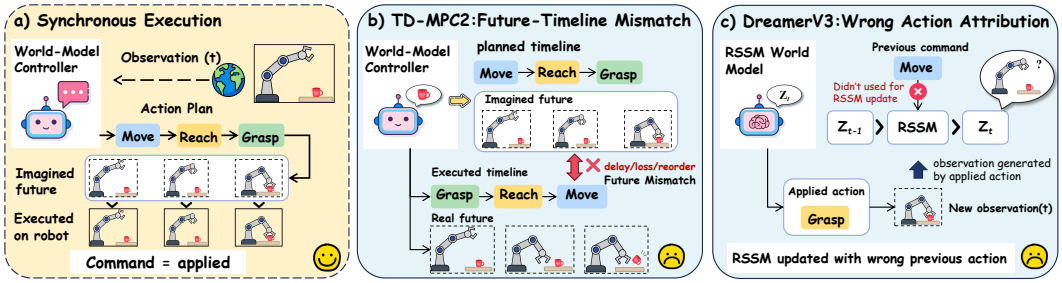


Figure 1: (a) Synchronous execution keeps commanded and applied actions aligned. Under asynchronous execution, (b) TD-MPC2 suffers from future-action timeline mismatch, while (c) DreamerV3 can suffer from previous-action attribution mismatch.

Execution-semantic gap. World-model control raises a different issue because actions serve both as outputs for execution and as explicit inputs to the learned predictive model. Existing delay-aware model-based methods mainly use learned dynamics to compensate for delayed interaction, while recent studies of asynchronous world models examine related misalignment on the observation side (Chen et al., 2021; Anand et al., 2026). Less well understood is how command–execution mismatch affects the action-conditioned computations inside a world-model controller when the action used by the model differs from the one that generates the corresponding environment transition. This effect depends on the architecture: planning-based controllers such as TD-MPC and TD-MPC2 roll future action sequences through learned latent dynamics, whereas recurrent world models such as DreamerV3 condition latent-state updates on previous actions (Hansen et al., 2022; 2024; Hafner et al., 2024). We therefore hypothesize that the same execution mismatch affects these architectures differently, producing a future-action timeline mismatch in planning-based world models and a previous-action attribution mismatch in recurrent world models.

Execution-consistent interfaces. Following latent-space MPC designs that roll candidate action sequences through learned dynamics, our ablations show that current applied-action or buffer information is insufficient for TD-MPC2; reliable planning instead depends on the future action timeline in the dynamics rollout (Figure 3). Inspired by packet-based communication and packetized predictive control in networked control systems, we introduce **Execution-Consistent Interfaces (ECI)**, a family of architecture-specific interfaces aligning world-model action inputs with actions realized by the execution system. For planning-based world models, we instantiate ECI as **Future-Sequence Execution**, which transmits the planner-generated future action suffix to an actuator-side buffer, preserving the action timeline considered during planning. For recurrent world models, we instantiate ECI as **Applied-Action Feedback**, which feeds the applied action into the recurrent state update, restoring the correct action–transition correspondence. Both interfaces modify only the controller–execution interface and leave the pretrained world model unchanged during deployment.

Key empirical findings. Our experiments reveal three consistent observations. **(1)** Command–applied divergence degrades control even with a fixed planned command sequence, and reliable latent planning requires the correct future action timeline inside the dynamics rollout. **(2)** Effective corrections differ across architectures: planning-based control benefits from preserving an updatable future action sequence, while recurrent control depends on timely action feedback consistent with training. **(3)** These effects extend across TD-MPC/TD-MPC2, multiple control domains, measured network-delay traces, and a process-separated asynchronous execution stack.

Contributions.Our main contributions are threefold. **(1)** We identify execution-semantic mismatch as a failure mode of world-model control and show that it manifests as a future-action timeline mismatch in planning-based TD-MPC2 and as a previous-action attribution mismatch in recurrent DreamerV3. **(2)** Guided by this diagnosis, we instantiate execution-consistent interfaces that preserve the planner's future action timeline for TD-MPC2 and feed back the actually applied action for DreamerV3, without modifying the underlying pretrained world models. **(3)** We validate these mechanisms across controller architectures, control tasks, delay, packet loss and reordering, measured network traces, and a process-separated asynchronous execution stack.

2 WHY DO WORLD-MODEL CONTROLLERS FAIL UNDER ASYNCHRONOUS EXECUTION?

In this section, we conduct a series of controlled empirical analyses to understand why world-model controllers degrade under asynchronous execution. Rather than treating delay, packet loss, or reordering merely as external disturbances, we focus on how these execution effects alter the action information consumed by the world model.

Preliminaries of asynchronous world-model control. At each control step t, the controller produces a command action u_t^c, while the execution system applies an applied action u_t^a to the environment. Under synchronous execution, $u_t^c = u_t^a$; under asynchronous execution, delay, packet loss, reordering, or buffering may instead lead to $u_t^c \neq u_t^a$. More generally, the applied action depends on the history of issued commands and the execution state:

$$u_t^a = g(u_{\leq t}^c, e_t), \tag{1}$$

where e_t denotes execution-side information such as packet arrivals, delays, and buffer contents. This distinction is important because world models are action-conditioned: planner-based controllers, such as TD-MPC and TD-MPC2, use future action sequences to roll latent states forward, whereas recurrent world models, such as DreamerV3, condition recurrent state updates on the previous action and the newly observed transition. As a result, asynchronous execution can induce different forms of action-semantic mismatch depending on where each architecture consumes action information. **Q1:** How does command–applied action divergence degrade planning-based world-model control, and what execution information is needed to correct it? **Q2:** Does execution-semantic mismatch manifest differently across world-model architectures?

2.1 FINDING 1: PLANNING-BASED WORLD MODELS NEED THE CORRECT FUTURE ACTION TIMELINE

We first isolate asynchronous execution from planner behavior by replaying the same planned command sequence from matched environment states while changing only the action-delivery delay. Figure 2 shows that greater delay increases command–applied divergence and reduces short-horizon return, even though the planner makes the same decisions. Thus, the execution process itself can degrade control, suggesting that TD-MPC2 may be missing information about how its commands will actually be realized.

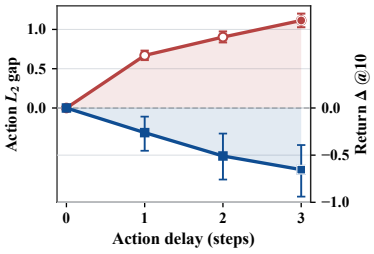


Figure 2: Effect of execution delay under fixed commands. Error bars denote 95% confidence intervals.

To identify this missing information, we progressively provide TD-MPC2 with the current applied action, the actuator buffer, or the exact future applied-action sequence. The first two describe only the current execution state, whereas the last specifies the action timeline that will actually drive the environment.

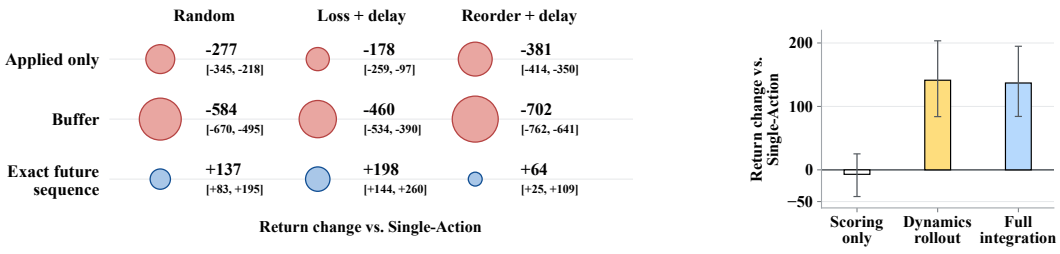


Figure 3: TD-MPC2 execution-information ablations. Left: information type across perturbations. Right: insertion point. Brackets and error bars denote 95% CIs.

Figure 3 shows that current applied-action or buffer information is insufficient, while the exact future sequence improves control only when it enters the latent dynamics rollout; using the same information for trajectory rescoring alone provides no benefit. This identifies the failure more precisely: TD-MPC2 predicts future states under one action sequence, while asynchronous execution may realize another. We refer to this as a *future-action timeline mismatch*. Reliable planning therefore requires the action timeline used inside the world-model rollout to remain consistent with the one realized by the execution system.

2.2 Finding 2: Execution-Semantic Mismatch Manifests Differently across World-Model Architectures

We next ask whether execution-semantic mismatch also appears in world-model architectures that consume actions differently. Unlike TD-MPC2, which uses future actions during latent rollout, DreamerV3 conditions its recurrent state update on the previous action and the new observation. Under asynchronous execution, the environment may evolve under one applied action while the recurrent model attributes the transition to a different issued command.

To isolate this effect, we vary only the previous-action input to DreamerV3's recurrent update while holding the observed trajectory and forecast action fixed. We evaluate both command-trained and applied-action-trained checkpoints, allowing us to test whether the benefit of applied-action feedback depends on the action semantics learned during training.

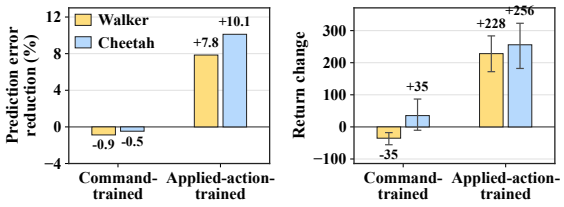


Figure 4: DreamerV3 feedback alignment under random delay.

Figure 4 links prediction and control: the left panel reports prediction-error reduction, while the right reports return change with 95% CIs. Applied-action feedback improves both measures for applied-action-trained checkpoints, but is not uniformly beneficial for command-trained models. This indicates that the recurrent-state update is sensitive not only to which action is supplied, but also to the action semantics learned during training. Together with Finding 1, this reveals two distinct failure modes: a future-action timeline mismatch in TD-MPC2 and a previous-action attribution mismatch in DreamerV3. Execution corrections should therefore target where each architecture consumes action information and remain consistent with its learned action semantics.

3 Execution-Consistent Interfaces

After identifying where asynchronous execution breaks the action semantics of world-model controllers, inspired by packetized transmission and receiver-side buffering in computer networks (Kurose & Ross, 2021), we introduce **E**xecution-**C**onsistent **I**nterfaces (ECI) , which restore the action information required by each architecture without modifying the pretrained world model. As illustrated in Figure 5, our approach consists of future-sequence execution for planning-based world models (Section 3.1) and applied-action feedback for recurrent world models (Section 3.2).

3.1 Future-Sequence Execution for Planning-Based World Models

Future-sequence transmission. At each control step t, TD-MPC2 produces a future action sequence over horizon H:

$$U_t^c = \left(u_{t|t}^c, u_{t+1|t}^c, \ldots, u_{t+H-1|t}^c\right), \tag{2}$$

where $u_{t+h|t}^c$ denotes the action planned at time t for time $t+h$. Under standard receding-horizon execution, only the current action $u_{t|t}^c$ is sent to the actuator. Under asynchronous execution, the next command can arrive too late because of transmission delay, be missing because of packet loss, or arrive out of order. The actuator then continues using actions from an earlier plan. To preserve the planner's future control information, we transmit the entire planner-derived sequence rather than only its first action. Each control packet contains the current action together with the

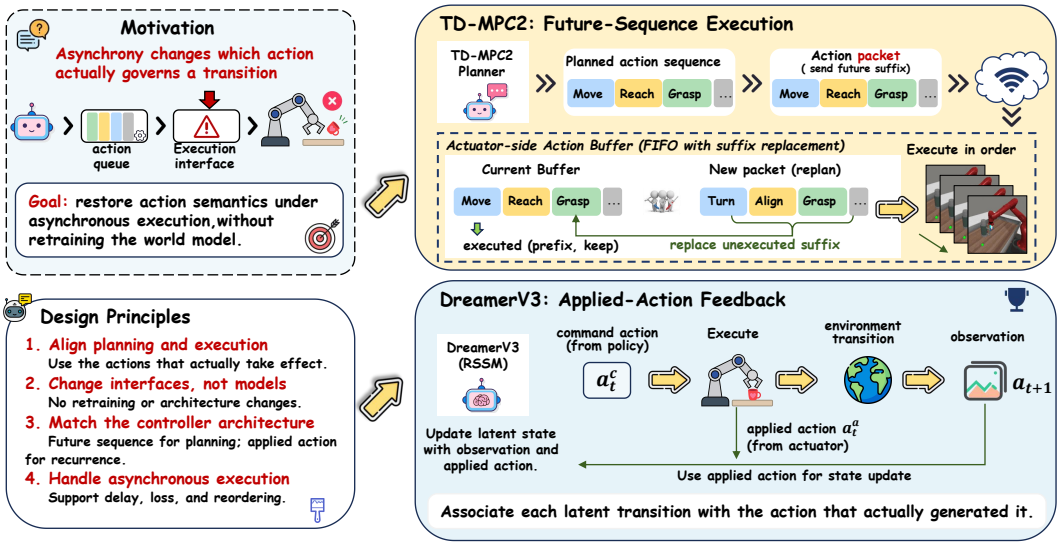


Figure 5: Overview of our execution-consistent world-model control framework.

remaining future actions and a generation index, providing the execution system with a ready-to-use continuation of the latest plan when subsequent commands are delayed, lost, or reordered.

Replaceable action buffer. Each transmitted sequence is associated with its planning time and stored at the actuator. Let τ_t denote the arrival time of the sequence generated at time t. At execution step k, the actuator uses the newest valid sequence that has already arrived and selects the action corresponding to the current step:

$$j_k = \max\{t \mid \tau_t \le k\}, \qquad u_k^a = \begin{cases} u_{k|j_k}^c, & 0 \le k - j_k < H, \\ u_k^{\mathrm{fb}}, & \text{otherwise}, \end{cases} \tag{3}$$

where u_k^{fb} denotes the fallback action when no valid planned action remains; we use a zero action in our experiments. When a newer sequence arrives, its unexecuted portion replaces the remaining actions from the previous sequence. Sequences that arrive after a newer plan has already been accepted are discarded, preventing stale plans from overwriting the current buffer.

Execution-consistent planning. The proposed interface does not predict future communication delays or introduce an additional control policy. Instead, it preserves future control information that TD-MPC2 has already produced. When a newly replanned command is unavailable, the actuator continues along the remaining actions of the newest available plan rather than immediately switching to an unrelated fallback action. This reduces the discrepancy between the future action sequence evaluated during world-model rollout and the sequence available for execution, directly addressing the future action timeline mismatch identified in Finding 1.

3.2 Applied-Action Feedback for Recurrent World Models

Applied-action feedback. DreamerV3 does not explicitly plan a future action sequence during online execution. Instead, its RSSM updates the recurrent state using the previous latent state, the previous action, and the current observation:

$$s_t = f_{\mathrm{RSSM}}(s_{t-1}, a_{t-1}, o_t). \tag{4}$$

Under synchronous execution, the issued command is also the action that generates the transition leading to o_t. Under asynchronous execution, however, the environment may actually evolve under a different applied action. If the RSSM still conditions on the issued command, it can associate the observed transition with the wrong action. We therefore replace the command action with the action that was actually applied:

$$s_t = f_{\mathrm{RSSM}}(s_{t-1}, u_{t-1}^a, o_t). \tag{5}$$

Table 1: Normalized TD-MPC2 return under different execution interfaces, averaged equally across tasks and normalized by matched clean SINGLE-ACTION return. Bold and underlined values mark the best and second-best TD-MPC2 results excluding the Oracle.

Method	Clean	Random	Bursty	Packet loss	Reordering	Adverse avg.
Execution baselines						
SINGLE-ACTION	1.000	0.305	0.215	0.232	0.358	0.278
HOLD-LAST	1.000	0.325	0.205	0.243	0.379	0.288
FIXED CHUNK	1.000	0.286	0.211	0.268	0.305	0.268
RECEDING BUFFER	1.000	**0.783**	**0.605**	<u>0.537</u>	**0.942**	<u>0.717</u>
FUTURE-SEQUENCE (ours)	1.000	**0.783**	**0.605**	**0.601**	**0.942**	**0.733**
External methods						
DATS	0.266	0.197	0.182	0.142	0.173	0.174
ACDA	0.407	0.458	0.430	0.422	0.433	0.436
Oracle diagnostic						
FUTURE-ACTION ORACLE	1.000	0.880	0.834	0.682	0.892	0.822

Execution receipt. To obtain the applied action without accessing simulator-internal information, the actuator returns an execution receipt together with the subsequent observation. The receipt identifies which action generated the transition, including its source packet and its position within that packet. It can either carry the applied-action vector directly or provide an identifier that allows the controller to recover the corresponding action from its local command history. Because the receipt is piggybacked on the observation, no additional communication round trip is required.

Correct action attribution. The purpose of this interface is not to provide DreamerV3 with future execution information, but to ensure that each observed transition is paired with the action that actually produced it. When the execution receipt arrives in time, the RSSM can update its recurrent state using the correct action, directly addressing the previous action attribution mismatch identified in Finding 2. The intervention changes only the information exchanged between the controller and the execution system, while leaving the pretrained world model and actor unchanged.

4 EXPERIMENTS

4.1 SETUP

Controllers and protocol. We study two structurally different world-model controllers, TD-MPC2 and DreamerV3. Unless explicitly studying delay-aware retraining, all pretrained models remain frozen during evaluation. We consider clean execution, fixed and random action delays, packet loss, and packet reordering, and further validate selected settings using measured network-delay traces and a process-separated asynchronous execution stack.

Benchmarks and metrics. Our primary experiments use the DeepMind Control Suite (Tassa et al., 2018): TD-MPC2 is evaluated on Dog, Walker, and Hopper, while DreamerV3 is evaluated on Walker, Cheetah, and Cup. Cross-domain experiments further cover MetaWorld (Yu et al., 2020), ManiSkill2 (Gu et al., 2023), and MyoSuite (Caggiano et al., 2022). We report performance using three key metrics: **episodic return** for overall control performance, **task success rate** for tasks with explicit completion criteria, and **command–applied action divergence** for quantifying execution-semantic mismatch. Deployment-oriented experiments additionally report per-step execution overhead and communication statistics. Primary results are reported with 95% confidence intervals.

Baselines. For TD-MPC2, we compare four execution-side baselines: SINGLE-ACTION sends only the current planned action; HOLD-LAST repeats the most recently available action when a new command is unavailable; FIXED CHUNK transmits and executes a fixed multi-action chunk; and RECEDING BUFFER maintains a buffered action sequence that is refreshed as new plans arrive. Our method, FUTURE-SEQUENCE, preserves the planner-generated future action sequence at the execution side, while a privileged FUTURE-ACTION ORACLE uses future applied actions only for diagnosis. We also compare with delay-aware learning methods, DATS (Chen et al., 2021) and

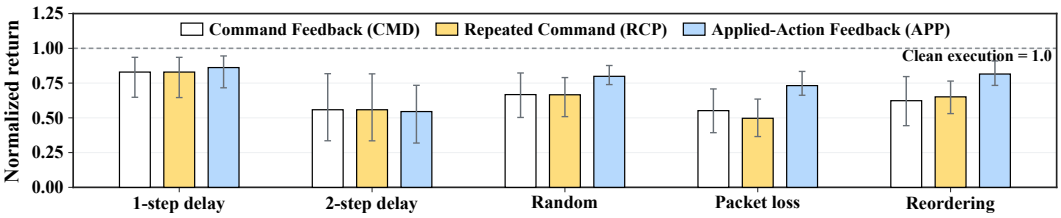


Figure 6: DreamerV3 under asynchronous execution. Each task is normalized by its mean clean-execution return under CMD and averaged equally across tasks; error bars denote 95% CIs.

ACDA (Wikman et al., 2026), which adapt the controller during delayed training. For DreamerV3, CMD conditions the recurrent update on the issued command, RCP repeatedly executes the current command, and our APP instead feeds back the actually applied action. ECT-FULL and ECT-ID implement APP in the process-separated stack using actuator-originated execution receipts.

4.2 EXPERIMENTAL RESULTS

Table 2: Future-Sequence return change on Walker.

Evaluation	Clean	Fixed-2	Random
Clean	+9	+10	+3
Fixed-1	+220	-324	-337
Fixed-2	-15	-473	-274
Fixed-3	-110	-343	-207
Random	+500	-28	+275
Loss + delay	+532	-19	+270
Reorder + delay	+525	-32	+294
Adverse avg.	**+276**	**-203**	**+3**

Cross-architecture results. Table 1 and Figure 6 summarize the main results on frozen TD-MPC2 and DreamerV3 controllers. For TD-MPC2, Future-Sequence preserves clean performance, substantially improves robustness across all adverse execution conditions, and achieves the highest reported adverse-condition average among the TD-MPC2 execution interfaces excluding the Oracle. It outperforms Single-Action, Hold-Last, and Fixed Chunk across all reported perturbations, matches Receding Buffer at the reported precision under random delay, bursty delay, and reordering, and achieves higher return under packet loss. For DreamerV3, Applied-Action Feedback provides the clearest gains under random delay, packet loss, and reordering, whereas Repeated Command offers little consistent benefit and fixed-delay improvements are less uniform. Together, these results support architecture-specific execution corrections: preserving the planner-generated future action sequence for planning-based world models, and correcting action attribution for recurrent world models. Additional numerical comparisons for DreamerV3 are provided in Appendix A. Figure 9 visualizes the difference within a paired Walker-Walk rollout under packet reordering. Both methods use the same checkpoint, initial state, environment seed, and transport seed. Single-Action loses balance early and attains a return of 397.4, whereas Future-Sequence remains upright throughout the episode and reaches 889.2. The replayed frames and body-height traces show how preserving a coherent future action sequence turns the aggregate return gain into sustained physical stability.

Generalization results. We further test Future-Sequence beyond the primary TD-MPC2 setting. It produces consistent gains on the original TD-MPC controller, across held-out MetaWorld, ManiSkill2, and MyoSuite tasks, and under measured network-delay profiles. These results show that the benefit is not restricted to TD-MPC2, a single task domain, or synthetically generated delay.

Deployment results. Figure 7 shows that the corrections remain effective when the controller and actuator run in separate processes communicating over UDP. Under injected delay, loss, and reordering, Future-Sequence recovers TD-MPC2 performance across the evaluated conditions, while Applied-Action Feedback improves DreamerV3 in most conditions. The required action feedback can be supplied directly by the actuator: ECT-FULL returns the applied-action vector, whereas ECT-ID identifies its source packet and position. Both recover the exact applied action across 30,000 tested transitions, demonstrating that action attribution can be implemented through execution receipts rather than privileged simulator access.

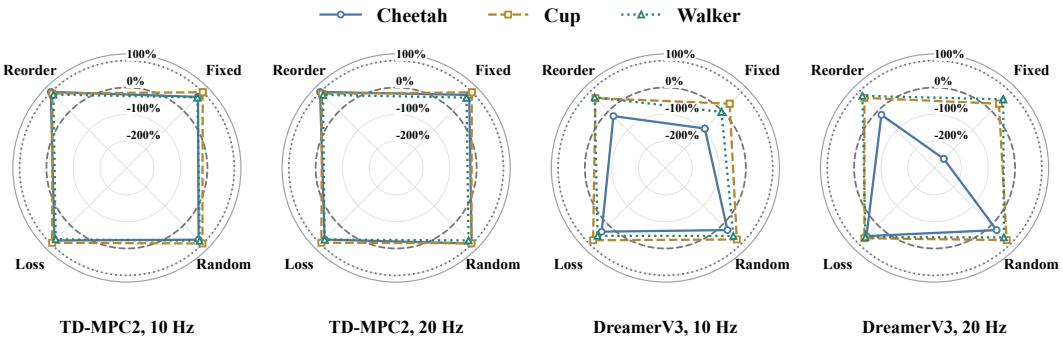


Figure 7: Lost return recovered in the asynchronous stack. From left: TD-MPC2 at 10/20 Hz, then DreamerV3 at 10/20 Hz. Dashed and dotted rings mark 0% and 100% recovery. All panels share a radial scale starting at -300%; negative values indicate further degradation.

4.3 Analysis

Training–interface interaction analysis. A natural question is whether Future-Sequence remains useful when the controller is already trained under delayed execution. Table 2 reveals a strong interaction. Clean-trained checkpoints benefit broadly, whereas combining Future-Sequence with Fixed-2 training degrades every adverse condition. Random-delay training is mixed: it improves stochastic delay, loss, and reordering but degrades fixed-delay conditions, leaving the adverse average nearly unchanged. This pattern shows that delay-aware training and execution-side correction are not automatically additive. The most plausible explanation is that delay-aware training changes the implicit action semantics of the controller, so an execution interface designed for a clean-trained controller can become mismatched or compensate twice. Absolute returns are reported in Appendix B.

Feedback-latency analysis. Figure 8 examines how feedback timeliness affects Applied-Action Feedback. At both evaluated control rates, immediate feedback yields the highest aggregate return, while one- and two-period feedback delays progressively reduce performance. Thus, access to applied actions alone is not sufficient: their temporal correspondence with the observed transitions also matters. This complements the action-semantics analysis in Section 2.2, identifying timely, transition-aligned feedback as an important requirement for effective recurrent-state correction.

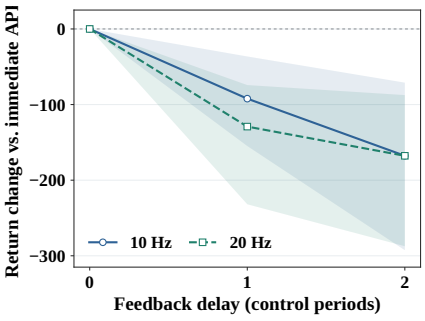


Figure 8: Feedback latency under random delay. Shading: 95% CI.

Execution-buffer analysis. Table 1 shows that preserving and updating a future action sequence outperforms transmitting isolated actions or executing fixed chunks. Both Future-Sequence and Receding Buffer substantially improve asynchronous control. They match at reported precision under random delay, bursty delay, and reordering, while Future-Sequence performs better under packet loss. Their implementations differ in updating the remaining sequence with incoming plans and handling execution after buffer exhaustion (Appendix D). Together with Figure 3, these results highlight the central role of retaining the planner's future action timeline, with buffer management further shaping robustness.

5 Related Work

5.1 Delayed reinforcement learning.

Delayed reinforcement learning studies decision making when observations or actions are delayed. Methods fall into three categories. The first augments state with pending actions or action histories to

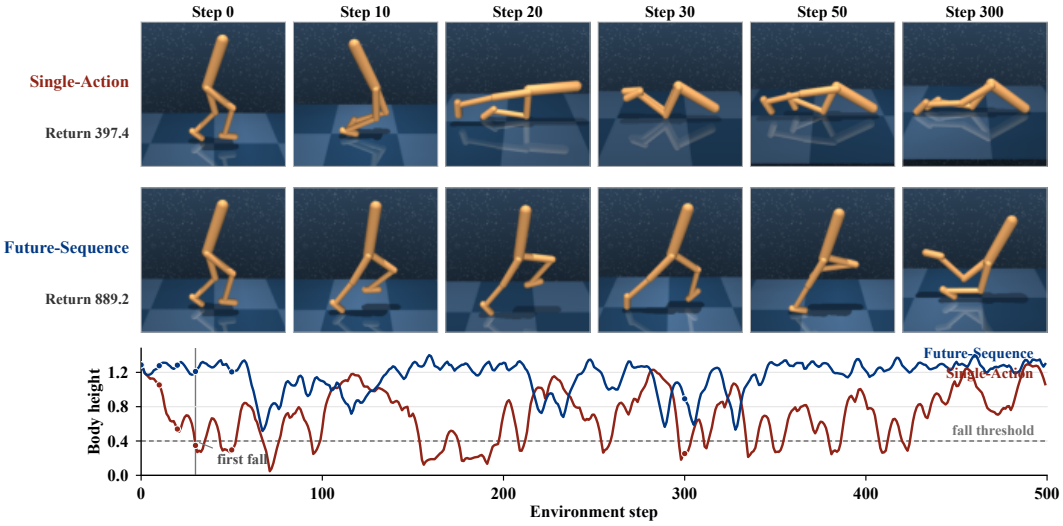


Figure 9: Paired Walker-Walk rollout under packet reordering.

recover the Markov property (Walsh et al., 2009; Derman et al., 2023). The second exploits delayed dynamics through state prediction or model-based reasoning, including real-time actor–critic, delay-aware model-based reinforcement learning, and belief-projection methods (Chen et al., 2021; Kim et al., 2023). The third addresses stochastic or partially observable delays through delay-corrected learning, latent planning, or adaptive interaction models (Bouteiller et al., 2021; Wu et al., 2024; Valensi et al., 2024; Wikman et al., 2026). Overall, this work progressed from representing delayed interactions to inferring decision-relevant states and actions under uncertain execution timing. Its primary objective is effective decision making under delay, not understanding how execution timing alters action-conditioned computations within a world-model controller.

5.2 World-model control under asynchronous execution.

World-model controllers use action-conditioned dynamics for prediction, policy optimization, and planning, including recurrent latent models such as PlaNet and Dreamer (Hafner et al., 2024), tree-search methods including MuZero and EfficientZero (Schrittwieser et al., 2020; Ye et al., 2021), and latent trajectory optimization in TD-MPC and TD-MPC2 (Hansen et al., 2022; 2024). World models handle delayed interaction via future-state prediction, delayed-observation imagination, and delay-aware planning (Wikman et al., 2026). These works mainly use world models to compensate for delay rather than study how asynchronous execution can invalidate assumed action semantics. Recent robot-learning methods emphasize prediction–execution alignment: RTC supports asynchronous action-chunk generation and execution, REMAC addresses inconsistencies between perception and concurrent actions, and FutureRTC studies prediction–execution misalignment (Black et al., 2025; Wang et al., 2026; Jiang et al., 2026); related work studies asynchronous sensor observations in world models (Anand et al., 2026). We therefore study the action side of world-model control under asynchronous execution and show how command–applied divergence induces future-action timeline mismatch in planning-based models and previous-action attribution mismatch in recurrent models.

6 Conclusion

In this paper, we identify how asynchronous execution disrupts action-conditioned computation in world-model controllers. Controlled interventions reveal a future-action timeline mismatch in TD-MPC2 and a training-dependent previous-action attribution mismatch in DreamerV3. Guided by these findings, execution-consistent interfaces improve control across tasks and asynchronous execution conditions. Our results highlight the importance of aligning execution-time action information with both model computation and learned action semantics.

AI USE STATEMENT

In this work, we used generative AI tools for language editing and research brainstorming. All AI-assisted text was reviewed and revised by the authors. AI-assisted research suggestions were manually checked against the relevant literature. We take full responsibility for the final content of this work, including all text and claims produced with the aid of generative AI.

ETHICS STATEMENT

This work studies the robustness of world-model controllers under asynchronous execution conditions. Our experiments are conducted in simulated control environments and do not involve human subjects, private data, or real-world deployment involving safety-critical decisions. We do not introduce new data collection procedures or applications that raise additional privacy, fairness, or security concerns. All experiments are performed for research purposes, and we take responsibility for the methods, analyses, and conclusions presented in this work.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of our problem formulation, execution-consistent interfaces, experimental protocols, and evaluation settings in the main paper and supplementary materials. The appendix includes additional implementation details, baseline configurations, ablation settings, and extended experimental results. We report the evaluated controllers, tasks, execution perturbations, and metrics to facilitate reproduction of our findings. Code and supplementary materials will be released to support independent verification and further research.

REFERENCES

Akash Anand, Abhay Anand, and Yash Vishe. World models under asynchronous sensor observations, 2026. URL https://arxiv.org/abs/2609.07299.

Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1122. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/300ccb2187dedd4edcc07f7e76d8e553-Abstract-Conference.html.

Yann Bouteiller, Simon Ramstedt, Giovanni Beltrame, Christopher Pal, and Jonathan Binas. Reinforcement learning with random delays. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=QFYnKlBJYR.

Carlos Stein Brito and Daniel McNamee. World models as reference trajectories for rapid motor adaptation. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=wyDg4x8FIY.

Vittorio Caggiano, Huawei Wang, Guillaume Durandau, Massimo Sartori, and Vikash Kumar. MyoSuite: A contact-rich simulation suite for musculoskeletal motor control. In Roya Firoozi, Negar Mehr, Esen Yel, Rika Antonova, Jeannette Bohg, Mac Schwager, and Mykel Kochenderfer (eds.), *Proceedings of The 4th Annual Learning for Dynamics and Control Conference*, volume 168 of *Proceedings of Machine Learning Research*, pp. 492–507. PMLR, 2022. URL https://proceedings.mlr.press/v168/caggiano22a.html.

Baiming Chen, Mengdi Xu, Liang Li, and Ding Zhao. Delay-aware model-based reinforcement learning for continuous control. *Neurocomputing*, 450:119–128, apr 2021. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2021.04.015. URL https://www.sciencedirect.com/science/article/pii/S0925231221005427.

Esther Derman, Gal Dalal, and Shie Mannor. Acting in delayed environments with non-stationary markov policies, 2023. URL https://arxiv.org/abs/2101.11992.

Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, Xiaodi Yuan, Pengwei Xie, Zhiao Huang, Rui Chen, and Hao Su. ManiSkill2: A unified benchmark for generalizable manipulation skills. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=b_CQDy9vrD1.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2024. URL https://arxiv.org/abs/2301.04104.

Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Oxh5CstDJU.

Nicklas A. Hansen, Hao Su, and Xiaolong Wang. Temporal difference learning for model predictive control. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 8387–8406. PMLR, 2022. URL https://proceedings.mlr.press/v162/hansen22a.html.

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

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.

Jangwon Kim, Hangyeol Kim, Jiwook Kang, Jongchan Baek, and Soohee Han. Belief projection-based reinforcement learning for environments with delayed feedback. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 678–696. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/0252a434b18962c94910c07cd9a7fecc-Paper-Conference.pdf.

James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. Pearson, 8 edition, 2021.

Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Fei Han, Mingrui Yu, Zelin Gao, Nan Xue, Xing Zhu, Yujun Shen, and Yinghao Xu. Causal world modeling for robot control, 2026. URL https://arxiv.org/abs/2601.21998.

Shangzhe Li, Zhiao Huang, and Hao Su. Reward-free world models for online imitation learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 34702–34724, 2025. URL https://proceedings.mlr.press/v267/li25af.html.

Pangun Park, Sinem Coleri Ergen, Carlo Fischione, Chenyang Lu, and Karl Henrik Johansson. Wireless network design for control systems: A survey. *IEEE Communications Surveys & Tutorials*, 20(2):978–1013, 2018. doi: 10.1109/COMST.2017.2780114. URL https://doi.org/10.1109/COMST.2017.2780114.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018. URL https://arxiv.org/abs/1801.00690.

David Valensi, Esther Derman, Shie Mannor, and Gal Dalal. Tree search-based policy optimization under stochastic execution delay. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Representation Learning*, volume 2024, pp. 18475–18495, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/50e13537f46656a94a7acaf022921385-Paper-Conference.pdf.

Thomas J Walsh, Ali Nouri, Lihong Li, and Michael L Littman. Learning and planning in environments with delayed feedback. *Autonomous Agents and Multi-Agent Systems*, 18:83–105, 2009.

Haoxuan Wang, Gengyu Zhang, Yan Yan, Yuzhang Shang, Ramana Rao Kompella, and Gaowen Liu. Real-time robot execution with masked action chunking. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2601.20130.

John Wikman, Alexandre Proutiere, and David Broman. Adaptive reinforcement learning for unobservable random delays. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR, 2026. URL https://arxiv.org/abs/2506.14411.

Philipp Wu, Alejandro Escontrela, Danijar Hafner, Pieter Abbeel, and Ken Goldberg. Daydreamer: World models for physical robot learning. In Karen Liu, Dana Kulic, and Jeff Ichnowski (eds.), *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 2226–2240. PMLR, 14–18 Dec 2023. URL https://proceedings.mlr.press/v205/wu23c.html.

Qingyuan Wu, Simon Sinong Zhan, Yixuan Wang, Yuhui Wang, Chung-Wei Lin, Chen Lv, Qi Zhu, and Chao Huang. Variational delayed policy optimization. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 54330–54356. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/61a18c8a7a1ea7445375dd7255905bc3-Paper-Conference.pdf.

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. October 2021.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura (eds.), *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pp. 1094–1100. PMLR, 2020. URL https://proceedings.mlr.press/v100/yu20a.html.

Lixian Zhang, Huijun Gao, and Okyay Kaynak. Network-induced constraints in networked control systems—a survey. *IEEE Transactions on Industrial Informatics*, 9(1):403–416, 2013. doi: 10.1109/TII.2012.2219540. URL https://doi.org/10.1109/TII.2012.2219540.

A DreamerV3 Numerical Results

Table 3 complements Figure 6 with the clean-normalized return improvement over Command Feedback. For each task, we divide the return difference by its mean clean-execution return under Command Feedback, then average equally across Walker, Cheetah, and Cup.

Table 3: DreamerV3 clean-normalized return improvement over Command Feedback. Means use ten evaluation seeds per task and equal task weights.

Method	Clean	1-step delay	2-step delay	Random	Packet loss	Reordering
Command Feedback	0.000	0.000	0.000	0.000	0.000	0.000
Repeated Command	0.000	0.000	0.000	-0.001	-0.055	0.028
Applied-Action Feedback	0.000	0.032	-0.013	**0.131**	**0.180**	**0.192**

B Delay-Aware Training Numerical Results

Table 4 provides the absolute returns underlying Table 2.

Table 4: Walker episodic return under different training and execution conditions. Entries average five independently trained checkpoints and ten evaluation seeds per checkpoint. F1 to F3 denote fixed delays, and Adv. denotes the mean over all non-clean conditions.

Training	Execution	Clean	F1	F2	F3	Random	Loss	Reorder	Adv.
Clean	SINGLE-ACTION	965	603	306	194	290	205	365	327
Clean	FUTURE-SEQUENCE	**974**	**824**	292	84	**790**	**737**	**890**	**603**
Fixed-2	SINGLE-ACTION	118	373	512	**379**	89	76	104	256
Fixed-2	FUTURE-SEQUENCE	129	49	39	36	61	58	72	52
Random 0–3	SINGLE-ACTION	896	812	**558**	337	313	248	384	442
Random 0–3	FUTURE-SEQUENCE	899	475	284	130	588	518	678	445

C IMPLEMENTATION DETAILS

Frozen evaluation. All execution-interface comparisons keep the controller parameters fixed. For each checkpoint, methods are paired using the same environment and transport seeds. At the beginning of each episode, the execution queue, held action, and controller state are reset. Controller training and execution-interface evaluation are conducted as separate experiments.

TD-MPC2 diagnostic planner. The execution-information and insertion-point ablations use a planning horizon of five, 64 candidate trajectories, eight policy-prior trajectories, and three base planning iterations. Following the original TD-MPC2 implementation, two additional iterations are used when the action dimension is at least 20. Returns are evaluated over 500-step episodes. During diagnostic evaluation, the planner's random-number state is preserved so that additional score queries do not alter subsequent planning samples.

DreamerV3 action alignment. The action supplied to the recurrent update is aligned with the transition that produces the current observation. Command Feedback uses the command issued for that transition, whereas Applied-Action Feedback uses the action actually applied to the environment. This action is distinct from the next command generated after observing the transition. The main DreamerV3 evaluation uses action repeat one and ten evaluation seeds per task and condition.

D BASELINE DEFINITIONS

TD-MPC2 interfaces. SINGLE-ACTION transmits only the first planned action. HOLD-LAST reuses the most recently available action when no new command arrives. FIXED CHUNK accepts a new action sequence only after its active queue is exhausted. RECEDING BUFFER accepts newer plans, discards actions whose scheduled execution times have passed, and replaces the remaining queue; after exhaustion, it holds the last applied action. Our FUTURE-SEQUENCE interface stores actions according to their intended execution slots, updates overlapping future slots when newer plans arrive, and rejects stale plans. The registered diagnostic implementation uses a zero-action fallback when no valid planned action remains.

Future-Action Oracle. The FUTURE-ACTION ORACLE evaluates candidate actions using an isolated copy of the network and actuator state, including future packet-delivery outcomes. The resulting applied-action sequence is then supplied to the world-model rollout. This Oracle is a privileged diagnostic condition rather than a deployable interface or a guaranteed upper bound on return.

DreamerV3 interfaces. Command Feedback (CMD) supplies the issued command to the recurrent update. Repeated Command (RCP) transmits five copies of the current command rather than a predicted future action sequence. Our Applied-Action Feedback (APP) instead supplies the actual previous action. In the process-separated implementation, ECT-FULL returns the applied-action vector, whereas ECT-ID returns its source packet and position, allowing the controller to reconstruct the action from its command history.

D.1 External Learning-Based Baselines

DATS. We reproduce DATS (Chen et al., 2021) using an ensemble of five probabilistic dynamics models, each with four hidden layers of width 200. The models predict transition means and variances and are trained with a learning rate of 10^{-3}, batch size 256, and five epochs per update. Planning uses CEM with horizon 30, 500 candidates, 50 elites, and five optimization iterations. Following DATS, pending actions are prepended to each candidate suffix so that planning accounts for actions already committed to execution. Training uses a fixed two-step action delay. For Walker, we use the analytic task reward; for Dog and Hopper, the learned model additionally predicts the observed reward.

ACDA. We adapt the released ACDA implementation (Wikman et al., 2026) to the DMC task interface and our runtime. ACDA uses a delay-indexed action matrix with horizon four, policy hidden layers of width 256, predictive-model hidden and recurrent dimensions of 256 and 384, and a model window of 16. The batch size is 256, replay capacity is 10^6, and the predictive-model learning rate is 10^{-4}. Delay memorization and model-output clamping are enabled. At execution time, ACDA produces its native delay-conditioned action matrix rather than a TD-MPC2 future sequence.

Evaluation protocol. DATS and ACDA train their own predictive models and controllers, whereas our Future-Sequence interface operates on a pretrained TD-MPC2 controller without modifying its parameters. We therefore evaluate each method as its complete controller–execution system. Checkpoints remain fixed during evaluation, and stochastic evaluations use ten environment seeds per checkpoint and condition with 500 control decisions per episode.

E Ablation Protocols

Fixed-command intervention. Figure 2 replays the same planned command sequence from matched environment states while varying action-delivery delay from zero to three steps. We measure command–applied action divergence and the change in ten-step return relative to zero-delay execution.

Execution-information ablation. Figure 3 compares three forms of execution information supplied during planning. Applied replaces only the first rollout action with the currently applied action. Buffer replaces rollout actions for time slots covered by the current actuator buffer while leaving the remaining candidate actions unchanged. Exact Future Sequence maps each candidate sequence to the actions produced by a copied execution system. The actual execution mechanism is held fixed across all information conditions. Each plotted contrast uses 20 paired episodes per perturbation.

Insertion-point ablation. Under random delay, Scoring Only supplies the mapped actions to the reward predictor while retaining command actions in the latent dynamics. Dynamics Rollout supplies mapped actions to the latent dynamics while retaining command actions in the reward predictor. Full Integration uses mapped actions in both components. All variants use the same terminal policy and value-estimation rule and are evaluated with 20 paired episodes. Full Integration therefore changes only where execution information enters the planner and does not involve additional model training.

Prediction and control alignment. Figure 4 evaluates Walker and Cheetah using command-trained and applied-action-trained checkpoints. For the prediction diagnostic, the observation trajectory and forecast action are fixed while only the previous-action input to the recurrent update is changed. Each checkpoint–task condition uses one held-out episode containing 950 evaluated prediction steps. Prediction improvement is defined as

$$100\frac{M_{\mathrm{CMD}} - M_{\mathrm{APP}}}{M_{\mathrm{CMD}}},$$

where M denotes observation-prediction MSE. Control improvement is evaluated separately using ten paired closed-loop episodes. Its 95% confidence interval is estimated with 20,000 paired bootstrap resamples, whereas prediction bars report descriptive means without confidence intervals.

Training–interface interaction. Table 2 compares clean training, fixed two-step-delay training, and random zero-to-three-step-delay training on Walker. Each training condition contains five checkpoints, each evaluated with ten seeds under both execution interfaces. The adverse average equally weights the six non-clean evaluation conditions. Differences are computed from unrounded returns before reporting rounded values.

Process-separated execution and feedback latency. The asynchronous execution stack runs at 10 and 20 Hz on Walker, Cheetah, and Cup, using five paired seeds per condition. Controlled execution shifts include a one-period fixed delay, uniformly sampled integer delays from zero to three periods, 20% packet loss combined with random delay, and packet reordering implemented by delaying selected packets by two periods with probability 0.60. The feedback-latency analysis uses random command delay and compares immediate, one-period-delayed, and two-period-delayed action feedback.

Metrics and aggregation. For the main DreamerV3 results, each task's return is normalized by its mean clean CMD return and then averaged equally across tasks. Appendix Table 3 instead reports the normalized return difference relative to CMD. Recovery in the process-separated stack is computed as

$$100\frac{R_{\mathrm{corrected}} - R_{\mathrm{baseline}}}{R_{\mathrm{natural}} - R_{\mathrm{baseline}}},$$

where natural execution uses the same baseline controller without injected perturbations. Return uncertainty is estimated using complete episodes as the sampling units rather than individual transitions.