T^{2}ALIGN: TEMPO AND TRANSITION ALIGNMENT FOR ASYNCHRONOUS ROBOT MANIPULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Chunked imitation learning policies, especially vision-language-action (VLA) models, have shown impressive performance in real-world robot manipulation by predicting temporally extended action chunks. However, deploying chunked policies in time-critical dynamic tasks remains challenging: task states change during execution, while success depends on reacting to these changes and completing motions within narrow temporal windows. We present T^{2}Align, a robot manipulation system that provides hierarchical tempo and transition alignment for asynchronous execution of chunked policies. At the execution level, T^{2}Align preserves the policy-intended execution tempo under safety constraints. At the action chunk level, T^{2}Align aligns returned chunks to the current execution timeline and stabilizes their transition through continuity-consistency blending and similarity-gated transition control. T^{2}Align is plug-and-play with off-the-shelf policies and improves their applicability to time-critical dynamic tasks. Experiments with widely used policies show that T^{2}Align substantially improves success rate and stability.

1 INTRODUCTION

Many imitation learning policies Zhao et al. (2023); Chi et al. (2025); Xue et al. (2025), including recent vision-language-action (VLA) Kim et al. (2024); Brohan et al. (2022); Zitkovich et al. (2023); Driess et al. (2023); Bjorck et al. (2025); Wen et al. (2025); Zhen et al. (2024), predict an *action chunk*—a temporally ordered sequence of future actions—to model complex trajectory and amortize inference. An action chunk encodes both spatial intent and temporal semantics. For the position-control policies considered here, the predicted targets specify where to move, while their indices and the demonstration sampling interval implicitly specify when trajectory progress should occur.

In static manipulation tasks with loose timing requirements, execution can prioritize reaching the correct spatial targets even when motions take longer than demonstrated. This tolerance does not extend to *time-critical dynamic tasks*, where task states change during execution and success depends on responding to these changes within narrow temporal windows. **Supporting such tasks is important for manipulation in changing real-world environments.** They impose two requirements: the robot must execute actions at an appropriate speed and react quickly to changed task states. An otherwise correct action can be ineffective if it occurs too early or too late (Fig. 8).

Meeting the execution-speed requirement calls for preserving *temporal semantics*: the nominal correspondence between policy progress and physical time induced by the demonstration sampling interval. Because policies produce actions in chunks, this correspondence must be maintained both *within* and *across* action chunks. Within a chunk, *execution tempo*—the rate of progress through original policy actions—should remain aligned with the *nominal rate* inherited from demonstration sampling. Across chunks, consecutive policy steps should retain the same nominal spacing rather than incur an additional pause at a chunk boundary. Synchronous execution introduces such a *boundary stall* when inference begins only after the current chunk finishes (Fig. 2(a)). Asynchronous inference can hide this stall by overlapping inference with ongoing execution (Black et al., 2025; Tang et al., 2025; Sendai et al., 2025). By launching inference earlier and switching to the returned chunk without waiting for the old chunk to finish, it can also reduce reaction latency (Fig. 2(b)).

However, preserving this correspondence faces two coupled challenges. First, *safety constraints distort within-chunk timing*. Prediction errors can produce an action whose displacement from the

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.

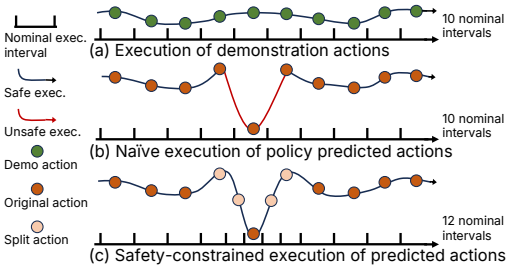

Figure 1: The nominal interval is the sampling interval of the demonstration data. Safety-constrained splitting preserves safe motion but slows the tempo of original policy actions.

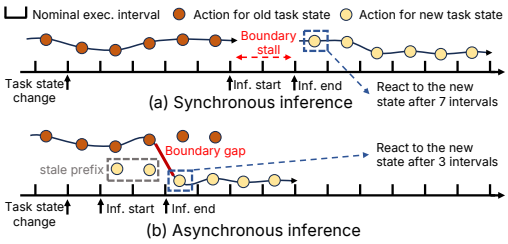

Figure 2: Synchronous and asynchronous inference after a task-state change. Asynchronous switching reacts earlier but introduces stale-prefix and boundary-gap problems.

previous target is too large to traverse within one nominal interval under the speed bound (Fig. 1(b)). The executor must allocate more time to this action (Fig. 1(c)). This slows progress through the original action sequence, reducing execution tempo below the nominal rate.

Second, *asynchronous inference complicates cross-chunk timing*. While inference runs, the robot continues executing the old chunk, so the observation conditioning the returned chunk is already stale at the transition. The new chunk may therefore contain a *stale prefix*: predictions for policy indices already covered by execution. This prefix must be discarded so that the new chunk continues from the next policy index. Discarding too few predictions repeats past indices causing rollback; discarding too many skips future ones. Even after correct prefix removal, predictions based on different observations can disagree at the transition, creating a *boundary gap* in action space (Fig. 2(b)). Such a gap can produce an abrupt motion or require additional time to traverse safely.

These challenges reinforce one another. Safety constraints make policy-action durations variable, so the number of actions completed during inference—and hence the stale-prefix length—can vary even when inference latency is unchanged. Estimating this length by dividing elapsed time by the nominal interval can therefore break policy-index continuity at a transition. Conversely, a large boundary gap can trigger the same safety constraint, introducing further timing deviation.

To address these coupled challenges, we present *T²Align*, a training-free asynchronous robot manipulation system for preserving temporal semantics within and across action chunks. *Execution-level tempo alignment* compensates for the extra time required by large-stride actions by executing safe small-stride actions faster, within system limits. It performs compensation online at the execution level, avoiding reliance on per-chunk compensation plans that may be interrupted by asynchronous chunk transition (Fig. 3(b)). *Chunk-level transition alignment* uses *stale-prefix alignment* based on logical timesteps to keep policy indices consecutive across transitions. After stale-prefix alignment, *continuity-consistency transition blending* reduces the boundary gap while maintaining consistency with the new chunk (Fig. 4(c)). *Similarity-gated transition control* disables this blending when adjacent chunks are directionally incompatible. Together, these mechanisms control accumulated timing deviation and stabilize chunk transitions without retraining the policy. We evaluate T^{2}Align with Diffusion Policy (DP) (Chi et al., 2025), ACT (Zhao et al., 2023), and $\pi_{0.5}$ (Intelligence et al., 2025) on three time-critical dynamic manipulation tasks, together with a diagnostic timing task.

In summary, we make four contributions. (1) We formalize the temporal semantics of action-chunked policies and identify how safety-constrained execution and asynchronous inference jointly disrupt timing within and across chunks. (2) We propose execution-level tempo alignment and chunk-level transition alignment to control accumulated timing deviation and maintain stable transitions. (3) We implement T^{2}Align as a training-free system for deploying off-the-shelf chunked policies. (4) We evaluate the system across representative policies and time-critical dynamic tasks, demonstrating improvements in task performance, timing accuracy, and execution stability.

2 Temporal Semantics of Action-Chunked Policies

2.1 Policy Time, Physical Time, and Temporal Semantics

We consider position-control policies trained on demonstrations sampled at a fixed rate f_{nom}, which we use as the *nominal rate* of policy progress. The corresponding *nominal interval* is

$\Delta t_{\text{nom}} = 1/f_{\text{nom}}$. This interval gives action indices a temporal meaning: consecutive position targets separated by distance s imply an average commanded speed of $s/\Delta t_{\text{nom}}$ when executed at the nominal rate. Thus, the demonstration sampling interval provides a timing reference for the predicted action sequence even though position targets do not explicitly specify velocities.

Policy time and physical time. We represent *policy time* by a logical timestep k, which counts completed original policy actions: after k actions have been completed, the next action has index k. An original action may be split into multiple low-level commands, but completing these commands together advances policy time by only one. *Physical time* t is elapsed wall-clock time, measured from the start of policy execution. *Execution tempo* is the rate at which policy time advances with physical time. Let $T(k)$ denote the physical time at which k original actions have been completed, with $T(0) = 0$. The physical duration of advancing one policy step is $\tau_k = T(k+1) - T(k)$.

Temporal semantics. The temporal semantics of an action sequence is the nominal correspondence between policy progress and physical time induced by the demonstration sampling interval:

$$T_{\text{nom}}(k) = k\Delta t_{\text{nom}}. \tag{1}$$

The realized correspondence is $T(k)$, and its accumulated timing deviation is

$$e(k) = T(k) - T_{\text{nom}}(k) = \sum_{i=0}^{k-1} \big(\tau_i - \Delta t_{\text{nom}}\big). \tag{2}$$

Positive $e(k)$ means execution is behind the nominal schedule. Our objective is to keep this deviation small subject to safety constraints .

2.2 Within-Chunk Timing under Safety Constraints

Within a predicted chunk, the nominal interval provides a reference time budget for each original policy action. However, safety constraints can require longer execution. Let $s_k = d_{\mathcal{A}}(a_{k-1}, a_k)$ be the *action stride* between consecutive position targets, where $d_{\mathcal{A}}$ is the action-space distance and a_{-1} denotes the initial target. Under a speed bound v_{safe}, the execution duration must satisfy

$$\tau_k \geq \frac{s_k}{v_{\text{safe}}}. \tag{3}$$

If $s_k/v_{\text{safe}} > \Delta t_{\text{nom}}$, executing this action safely necessarily increases timing deviation:

$$e(k+1) - e(k) = \tau_k - \Delta t_{\text{nom}} > 0. \tag{4}$$

2.3 Cross-Chunk Timing under Policy Inference

Across chunks, the nominal interval remains the demonstration sampling interval for consecutive actions. A transition should not introduce an additional pause solely because of inference latency. It must also continue from the next policy index, rather than repeat past indices or skip future ones.

Synchronous inference introduces a boundary stall. If inference begins only after the current chunk finishes, physical time advances while policy progress waits. With inference latency L, and assuming the next action itself takes one nominal interval, the boundary interval becomes

$$\tau_k = L + \Delta t_{\text{nom}}. \tag{5}$$

This additional waiting time is a *boundary stall* (Fig. 2(a)). Asynchronous inference can hide it by overlapping inference with execution. Launching inference earlier can also reduce reaction latency.

Asynchronous inference requires consecutive policy indices. Consider an observation captured at policy-step boundary k_s. The resulting chunk is $A^{\text{new}} = [a_0^{\text{new}}, \ldots, a_{H-1}^{\text{new}}]$, where H is the chunk horizon and local offset h corresponds to global policy index $k_s + h$. Because execution continues during inference, the conditioning observation is already stale when the new chunk becomes available. At the transition, let k_c be the next policy index to execute. The old chunk has already covered $d = k_c - k_s$ steps since the observation. The new chunk therefore has a *stale prefix*: predictions corresponding to these already covered indices. To preserve policy-index continuity, the system must

discard this prefix and continue from offset d. This requires $d < H$; otherwise, the returned chunk has no remaining action at the current policy index.

A latency-based estimate uses $\hat{d} = \lfloor L/\Delta t_{\text{nom}} \rfloor$, where L here is the elapsed physical time from the conditioning observation to the transition, including inference and any additional scheduling delay. This estimate assumes nominal-rate policy progress. Under safety constraints, action durations vary, so the number of completed actions can vary even for the same L. An incorrect estimate produces an index mismatch. Discarding too few predictions repeats previously covered indices, causing rollback; discarding too many skips future ones.

3 METHOD

T^{2}Align addresses the challenges in Sec. 1 through two coupled mechanisms. *Execution-level tempo alignment* allocates time to individual policy actions to control accumulated timing deviation. *Chunk-level transition alignment* maintains consecutive policy indices and stabilizes how a returned chunk replaces the current chunk. The system architecture is described in Appendix A.1.

3.1 EXECUTION-LEVEL TEMPO ALIGNMENT

Actions that require more than one nominal interval to satisfy the speed constraint slow policy progress. We compensate for this extra time using actions that can be executed safely in less than one nominal interval. Planning compensation independently within each predicted chunk is insufficient: asynchronous replacement may discard the compensating actions before they are executed (Fig. 3(a)). We therefore allocate time online at the execution level and maintain a time ledger.

Time ledger. Following Sec. 2, let k count completed policy actions and let $T(k)$ be their completion time. We maintain

$$B_k = k\Delta t_{\text{nom}} - T(k) = -e(k) \tag{6}$$

A negative ledger indicates that execution is behind the nominal schedule; a positive ledger represents time saved by earlier execution. Upon completing action a_k, we update the ledger using the elapsed duration $\tau_k = T(k+1) - T(k)$:

$$B_{k+1} = B_k + \Delta t_{\text{nom}} - \tau_k. \tag{7}$$

Extra execution time reduces the ledger, while completing a policy step in less than one nominal interval increases it. The ledger records execution progress relative to the nominal schedule.

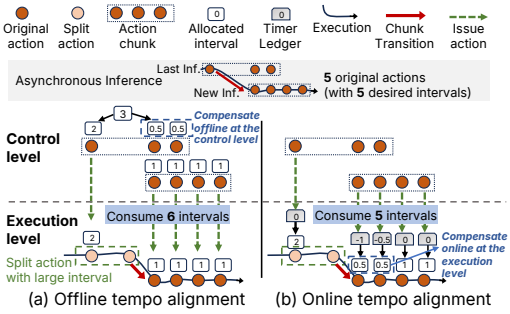

Figure 3: Execution-level tempo alignment. (a) A per-chunk compensation plan may be interrupted by asynchronous chunk replacement. (b) A persistent time ledger allows compensation to continue through the actions actually executed, including those from a replacement chunk.

Online time allocation. Let a_k be the action predicted for policy step k. Its stride is $s_k = d_{\mathcal{A}}(a_{k-1}, a_k)$, and the speed bound requires a duration of at least $\tau_k^{\text{req}} = max(s_k/v_{\text{safe}}, \tau_{min})$, where τ_{min} is determined by the robot maximum control frequency. We allocate

$$\tau_k^{\text{alloc}} = \begin{cases} \tau_k^{\text{req}}, & \tau_k^{\text{req}} \geq \Delta t_{\text{nom}}, \\ \Delta t_{\text{nom}} - g(B_k)\big(\Delta t_{\text{nom}} - \tau_k^{\text{req}}\big), & \tau_k^{\text{req}} < \Delta t_{\text{nom}}, \end{cases} \tag{8}$$

where

$$g(B) = \begin{cases} 1, & B \leq 0, \\ \max(0, 1 - B/B_{\text{ref}}), & B > 0. \end{cases} \tag{9}$$

The positive parameter B_{ref} bounds how much positive slack the system can accumulate. If an action requires extra time, the executor allocates the required duration. Otherwise, the available slack is used in proportion to $g(B_k)$. When execution is behind schedule, the full available slack is used for compensation. Once execution is ahead, the gain decreases and reaches zero at B_{ref}. Allowing positive credit lets earlier small-stride actions absorb some later slowdowns without requiring knowledge

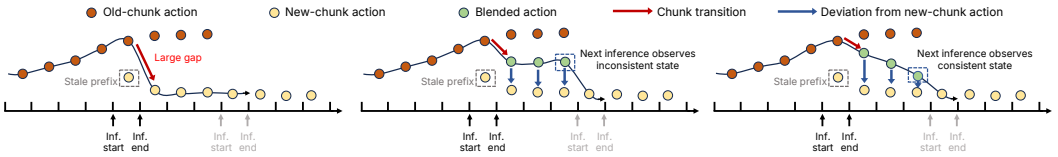

Figure 4: Continuity-consistency transition blending after stale-prefix alignment. (a) Direct switching can introduce a large boundary gap. (b) Continuity-only blending can retain excessive influence from the old chunk. (c) Increasing the weight on the new chunk gradually shifts execution toward its updated response while retaining the old continuation near the start of the transition.

of future chunks. To maintain high-frequency command issuing, a large-stride action is interpolated into multiple smaller commands over its allocated duration. Completing these commands advances policy time only by one. Chunk replacement does not reset the ledger, so compensation can continue in the new chunk. This aligns execution tempo at the level of policy progress.

3.2 Chunk-Level Transition Alignment

Chunk-level transition alignment addresses the timing and transition problems introduced by asynchronous inference. Stale-prefix alignment establishes index continuity; transition blending and similarity gating address the boundary gap that can remain after this alignment.

3.2.1 Stale-Prefix Alignment

Instead of modeling asynchronous inference in continuous time, we model it in discrete timesteps. Whenever the system *consumes* one original action, the logical timestep increases by 1. When an observation is captured, we record its logical timestep k_s. The resulting chunk $A^{\text{new}} = [a_0^{\text{new}}, \ldots, a_{H-1}^{\text{new}}]$ associates local offset h with global policy index $k_s + h$. At the transition, let k_c be the next policy index to execute. We discard $d = k_c - k_s$ predictions and retain $[a_d^{\text{new}}, \ldots, a_{H-1}^{\text{new}}]$, provided $d < H$. This uses actual policy progress rather than estimating it from profiled inference latency and profiled per-action execution latency.

3.2.2 Continuity-Consistency Transition Blending

After stale-prefix alignment, the system must then transition from the old chunk to the new chunk. However, a large boundary gap between adjacent chunks reduces motion continuity and can harm execution tempo (Fig. 4(a)). Blending their overlapping predictions can improve continuity, but smoothing alone may retain excessive influence from the old chunk and delay the updated response (Fig. 4(b)). We therefore consider both *continuity* at the transition and *consistency* with the new prediction at each policy index. Consistency matters to subsequent inference as well: When blended actions deviate from the new predictions, the resulting observations may reflect states that differ from those implied by the new chunk.

Let M be the number of remaining policy indices covered by both the old chunk and the new chunk. For $j = 0, \ldots, M-1$, let o_j and $u_j = a_{d+j}^{\text{new}}$ be the corresponding actions at global index $k_c + j$ of the old chunk and the new chunk, respectively. For Cartesian action dimensions, we use

$$b_j = (1 - \lambda_j) o_j + \lambda_j u_j, \qquad \lambda_j = \frac{j+1}{M+1}. \tag{10}$$

Rotational targets are interpolated using the same weights. The increasing weights gradually transfer influence from the old continuation to the new predictions (Fig. 4(c)).

3.2.3 Similarity-Gated Transition Control

Blending can be harmful when the two chunks encode incompatible motion directions, producing an intermediate continuation that follows neither response. We therefore gate blending using directional similarity. Let a be the transition anchor, chosen as the latest issued action, and let o_{end} and u_{end} be the last targets in the old and new chunks, respectively. We compute

$$\mathbf{d}_{\text{old}} = o_{\text{end}} - a, \qquad \mathbf{d}_{\text{new}} = u_{\text{end}} - a, \qquad \rho = \frac{\mathbf{d}_{\text{old}}^{\top} \mathbf{d}_{\text{new}}}{\|\mathbf{d}_{\text{old}}\|_2 \|\mathbf{d}_{\text{new}}\|_2}. \tag{11}$$

When an overlap exists and $\rho \geq \rho_{\min}$, where $\rho_{\min}$ is the similarity threshold, we apply the blending above. Otherwise, we use the aligned new suffix directly. This prioritizes reacting to the latest observation when the old and new chunks disagree.

4 Evaluation

4.1 Experimental Setup

Policies and settings. We evaluate Diffusion Policy (DP), ACT, and $\pi_{0.5}$. For real-world tasks, the chunk horizon is $H = 48$ for Whack-A-Mole and Cart Gate, and $H = 96$ for Press Car and Press Timer. Unless otherwise specified, the asynchronous inference is launched after $H/2$ actions of the current chunk are issued and $B_{\text{ref}} = 0.2\,\text{s}$. Hardware details are in Appendix C.1, and **demonstration videos are provided in the supplementary material.**

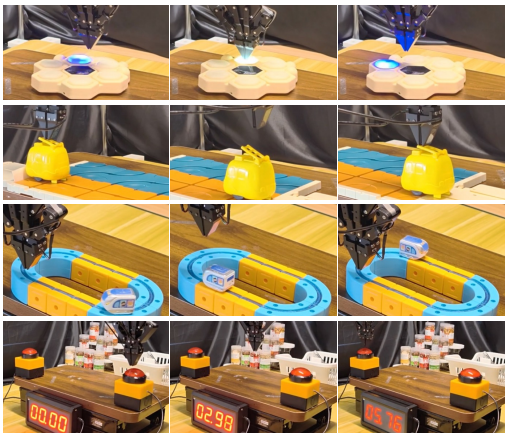

Figure 5: Illustration of the experimental tasks: Whack-A-Mole, Press Car, Cart Gate, and Press Timer. **Demo videos are provided in the supplementary material**.

Tasks. Our real-world evaluation comprises three time-critical dynamic tasks and one diagnostic timing task (Fig. 5). Demonstrations for these tasks are collected by kinesthetic teaching. Each policy–deployment configuration is evaluated for 50 episodes under the same task setup. **Whack-A-Mole** requires pressing the randomly illuminated target among six buttons before it switches after 2 s. **Press Car** requires two presses on a cart's direction-changing button: first to activate the cart, then to reverse it before it enters a failure region. **Cart Gate** requires lifting the arm from a circular track before the cart arrives and lowering it immediately after the cart passes; collisions or remaining raised count as failures. **Press Timer** diagnoses execution tempo using a start–stop interval demonstrated within $[4.8, 5.2]\,\text{s}$. Details are provided in Appendix C.2.

Metrics. We report correct hits per 30 s (Hits) for Whack-A-Mole and success rate (Succ.) for Press Car and Cart Gate. For Press Timer, Deviation is $\max(0, 4.8 - t_{\text{dur}}, t_{\text{dur}} - 5.2)$, where t_{dur} is the start–stop duration in seconds. Tremor counts sequences of at least two consecutive back-and-forth motions without meaningful task progress. Translational and rotational jerk $(J_{\text{xyz}}, J_{\text{rpy}})$ measure motion smoothness. For end-effector samples $\mathbf{x}_n$ at interval Δt,

$$\mathbf{j}_n = \frac{\mathbf{x}_{n+3} - 3\mathbf{x}_{n+2} + 3\mathbf{x}_{n+1} - \mathbf{x}_n}{\Delta t^3}. \tag{12}$$

4.2 Real-World End-to-End Evaluation

Table 1 compares the original synchronous and naive asynchronous deployments (baselines) with T^{2}Align (Ours). Sync (Ours) enables tempo alignment only; Async (Ours) enables both tempo and transition alignment. Naive asynchronous execution performs worse than synchronous execution on Whack-A-Mole and Press Car because the unresolved boundary gaps disturb motions. Tempo alignment improves synchronous performance on Whack-A-Mole and Press Car. Full asynchronous T^{2}Align further improves task performance over Sync (Ours) across all three dynamic tasks, supporting the combination of tempo control with aligned and stable chunk transitions.

4.3 Comparison with Asynchronous Inference Baselines

Task and protocol. We construct *Timer Whack-A-Mole* in MuJoCo simulation by combining the changing targets of Whack-A-Mole with the timing requirement of Press Timer (Sec. 4.1). This task evaluates hit-time accuracy under inference latency and execution speed constraints. The environment has five targets, with one active at a time for up to 1.5 s, a desired hit time of 1.1 s after

Table 1: End-to-end performance on dynamic manipulation tasks. For Whack-A-Mole, Hits denotes the number of correct hits within 30 seconds, and Tremor counts tremor events caused by unstable repeated motions. For Press Car and Cart Gate, Succ. denotes the success rate. J_{xyz} and J_{rpy} denote translational and rotational jerk, respectively.

Policy	Method	Whack-A-Mole				Press Car			Cart Gate		
		Hits ↑	Tremor ↓	J_{xyz} ↓	J_{rpy} ↓	Succ. ↑	J_{xyz} ↓	J_{rpy} ↓	Succ. ↑	J_{xyz} ↓	J_{rpy} ↓
DP	Sync (baseline)	4.16	–	**0.715**	4.262	50%	0.328	2.376	52%	**0.478**	0.827
	Sync (Ours)	6.84	–	0.760	**3.111**	64%	0.504	2.218	50%	0.651	0.819
	Async (baseline)	2.16	2.50	6.292	6.494	0%	1.026	5.644	62%	2.465	1.082
	Async (Ours)	**12.16**	**0.50**	0.829	3.458	**76%**	**0.094**	**1.282**	**88%**	0.692	**0.285**
ACT	Sync (baseline)	4.02	–	0.846	5.966	24%	0.402	0.808	34%	0.145	**0.004**
	Sync (Ours)	9.14	–	1.029	**2.938**	60%	0.304	**0.111**	32%	0.156	**0.004**
	Async (baseline)	1.44	4.28	3.261	13.011	0%	2.230	5.446	58%	1.570	0.057
	Async (Ours)	**11.96**	**0.30**	**0.645**	5.099	**80%**	**0.123**	0.326	**82%**	**0.120**	0.006
PI0.5	Sync (baseline)	1.04	–	1.101	2.659	0%	1.029	1.680	32%	0.503	0.027
	Sync (Ours)	3.18	–	1.143	2.834	82%	1.054	1.124	36%	0.531	0.028
	Async (baseline)	0.84	4.50	6.816	11.116	0%	1.633	1.878	54%	1.490	0.077
	Async (Ours)	**10.16**	**0.66**	**0.763**	**1.544**	**96%**	**0.672**	**0.058**	**94%**	**0.496**	**0.024**

Table 2: Comparison on Timer Whack-A-Mole using the same PI0.5 checkpoint. Deviation is in seconds.

Method	Deviation ↓	Hits ↑	Jerk ↓
Sync	0.2061	1.10	6.461
RT-C	0.1996	17.74	7.403
PAINT	0.3286	0.34	6.765
VLA-RAIL	0.1434	13.90	**3.924**
T^{2}Align	**0.0924**	**18.90**	4.555

Table 3: Effect of execution-level tempo alignment. Deviation is measured on Press Timer; all other metrics are measured on Whack-A-Mole.

Policy	Tempo	Deviation ↓	Hits ↑	Tremor ↓	J_{xyz} ↓	J_{rpy} ↓
DP	Disabled	1.039	2.44	2.42	4.854	5.351
	Enabled	**0.006**	**12.16**	**0.50**	**0.829**	**3.458**
ACT	Disabled	0.516	1.26	1.50	5.929	11.276
	Enabled	**0.000**	**11.96**	**0.30**	**0.645**	**5.099**
PI0.5	Disabled	1.284	2.34	4.22	6.616	9.351
	Enabled	**0.674**	**10.16**	**0.66**	**0.763**	**1.544**

activation, and a 0.3 s blank period between target activations. All methods share a PI0.5 checkpoint trained on 600 demonstration trajectories, whose hit times are 1.0814 ± 0.0918 s (mean $\pm$ standard deviation). The nominal action rate is 30 Hz, the chunk horizon is $H = 24$, and each episode lasts 30 s of task time. For a recorded correct hit, timing deviation is $|t_{\text{hit}} - 1.1\,\text{s}|$, where t_{hit} is measured from target activation. We report this timing deviation (*Deviation*), hit count per 30 s (*Hits*), and motion jerk. For T^{2}Align, the B_{ref} parameter in execution-level tempo alignment is set to 0.1 s.

Results. Table 2 shows that T^{2}Align achieves the lowest reported timing deviation and the highest hit count. VLA-RAIL achieves the lowest jerk, whereas T^{2}Align has the second-lowest jerk alongside better hit timing and a higher hit count. Thus, the smoothest motion does not coincide with the best timing accuracy or task progress in this setting. PAINT introduces additional inference overhead increasing inference latency from 291.3 ms to 815.5 ms on Nvidia A100, and causing it to frequently miss the hit timing, resulting in only 0.34 hits, close to the performance of synchronous inference. Taken together, these results show that T^{2}Align improves hit timing and task progress while retaining relatively smooth motion.

4.4 Component Analysis

In the following, we evaluate the effect of each component of T^{2}Align. Unless otherwise specified, all components are enabled by default.

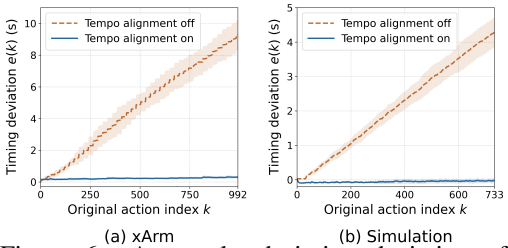

Figure 6: Accumulated timing deviation of PI0.5 on Whack-a-Mole.

Effect of execution-level tempo alignment. We evaluate the effect of execution-level tempo alignment on Press Timer and Whack-A-Mole. The results are reported in Table 3. The *Enabled* setting enables execution-level tempo alignment, while the *Disabled* setting removes it. For Whack-A-Mole, we use asynchronous inference with chunk-level transition alignment enabled. For Press Timer, we use synchronous inference, because this task isolates whether the system preserves the policy-intended execution tempo. Execution-level tempo alignment substantially reduces the Press Timer deviation across all three policies, showing that it helps preserve the timing implied by the demonstrations. On Whack-A-Mole, it also improves

Table 4: Effect of continuity-consistency transition blending on Whack-A-Mole. None denotes direct switching without transition blending. Uniform and RT-C are continuity-only blending methods, while Ramped is our continuity-consistency blending method.

Policy	Method	Hits ↑	Tremor ↓	J_{xyz} ↓	J_{rpy} ↓
DP	None	7.84	4.12	3.881	10.143
	Uniform	8.98	3.04	5.167	15.020
	Ramped	**12.04**	**2.00**	**0.941**	**5.024**
ACT	None	4.82	3.40	3.153	7.903
	Uniform	7.40	1.40	3.461	7.982
	Ramped	**10.40**	**0.60**	**0.842**	**5.410**
PI0.5	None	5.58	6.02	5.415	7.364
	Uniform	5.60	7.66	6.105	12.896
	RT-C	7.14	6.98	4.307	9.564
	Ramped	**10.10**	**2.06**	**1.168**	**3.768**

Table 5: Success rate of PI0.5 using different stale-prefix alignment strategies.

Method	LIBERO-10	Goal	Object	Spatial	All
None	42.4%	72.0%	57.2%	34.4%	51.5%
Profiled-Latency Align.	47.0%	79.2%	66.6%	37.0%	57.5%
Timestep Align.	**94.8%**	**92.0%**	**96.2%**	**97.6%**	**95.2%**

Table 6: Effect of similarity-gated transition control on Whack-A-Mole.

Policy	Gate	Hits ↑	Tremor ↓	J_{xyz} ↓	J_{rpy} ↓
DP	Disabled	12.04	2.00	0.941	5.024
	Enabled	**12.16**	**0.50**	**0.829**	**3.458**
ACT	Disabled	10.40	0.60	0.842	5.410
	Enabled	**11.96**	**0.30**	**0.645**	**5.099**
PI0.5	Disabled	10.10	2.06	1.168	3.768
	Enabled	**10.16**	**0.66**	**0.763**	**1.544**

the number of correct hits and reduces tremor and jerk. This improvement comes from preserving the progress rate of the original policy action sequence. We further report the accumulated timing deviation $e(k)$ (Eq. 2) during the Whack-a-Mole experiments on the real xArm robot and in the MuJoCo simulator (Fig. 6). Tempo alignment substantially reduces the accumulation of timing deviation, supporting improved alignment between policy progress and physical time.

Effect of stale-prefix chunk alignment. We replay execution-latency patterns measured on real-world Whack-A-Mole in LIBERO and evaluate $\pi_{0.5}$ on four suites of ten tasks, with 50 episodes per task. Logical-timestep-based stale-prefix alignment achieves 95.2% overall success, compared with 57.5% for profiled-latency alignment and 51.5% without alignment (Table 5). The improvement across all suites supports using actual policy progress rather than profiled durations for cross-chunk indexing under variable execution times.

Effect of continuity-consistency transition blending. With the similarity gate disabled, we compare direct switching (None), fixed-weight blending (Uniform), and our continuity-consistency blending on asynchronous Whack-A-Mole; RT-C is additionally evaluated on $\pi_{0.5}$ (Table 4). Our blending achieves the highest Hits and lowest Tremor, J_{xyz}, and J_{rpy} for each policy. Uniform blending does not consistently improve stability over direct switching, and RT-C retains frequent tremor despite increasing hits. These results support gradually shifting the transition toward the new chunk rather than retaining fixed influence from the old predictions.

Effect of similarity-gated transition control. We evaluate the effect of similarity-gated transition control on Whack-A-Mole. All settings use Ramped Transition Blending, and we compare whether the transition gate is enabled. Enabling the gate substantially reduces tremor and jerk across all policies, indicating better execution stability (Table 6). Although continuity-consistency blending improves most chunk transitions, blending can be harmful when the old queued chunk and the new chunk encode substantially different intents. Similarity-gated transition control detects such low-similarity transitions and disables blending when the chunk intents are incompatible. Therefore, it reduces harmful blended actions and makes transition more robust under asynchronous execution.

4.5 Cross-Hardware Applicability

Table 7: Cross-hardware applicability of T^{2}Align.

Setting	Sync (Base.)		Sync (Ours)		Async (Base.)		Async (Ours)	
	Succ. ↑	Jerk ↓	Succ. ↑	Jerk ↓	Succ. ↑	Jerk ↓	Succ. ↑	Jerk ↓
Cart Gate	22%	0.781	28%	0.741	44%	4.664	**92%**	**0.417**

Cross-hardware applicability of T^{2}Align. We further evaluate T^{2}Align on a different robot platform, the Flexiv Rizon 4s. We use PI0.5 on the Cart Gate task. As shown in Table 7, asynchronous inference with T^{2}Align achieves the highest success rate and the lowest jerk among all compared settings. These results indicate that T^{2}Align remains effective beyond the xArm 7 platform.

5 Related Work

5.1 Chunked Imitation Learning Policies

Action chunking Chi et al. (2025); Zhao et al. (2023); Ze et al. (2024); Zhang et al. (2025); Høeg et al. (2024); Jiang et al. (2025); Xue et al. (2025) has become a common design choice for imitation learning policies, as it improves the modeling of complex trajectories and long-horizon dependencies. Recent VLA policies Kim et al. (2025); Black et al. (2024); Intelligence et al. (2025); Liu et al. (2024); Cheang et al. (2024); Li et al. (2024); Bjorck et al. (2025); Wen et al. (2025); Shukor et al. (2025); Hou et al. (2025); Hung et al. (2025); Song et al. (2025); Zhao et al. (2025a) adopt action chunks and show strong physical manipulation performance. By predicting multiple actions in one forward pass, action chunking amortizes policy inference and reduces the frequency of expensive neural inference. However, during the execution of a chunk, the policy cannot react to state changes. This increases reaction latency and becomes harmful in time-critical dynamic tasks.

5.2 Asynchronous Inference for Chunked Policies

Recent works use asynchronous inference to overlap policy inference with chunk execution Black et al. (2025); Shukor et al. (2025); Xue et al. (2025); Sendai et al. (2025); Tang et al. (2025); Ho et al. (2026); Zhao et al. (2025b); Wang et al. (2026), reducing the execution stall caused by synchronous inference. Several approaches improve asynchronous execution without retraining the base policy. RT-C Black et al. (2025) improves chunk-level continuity for flow-matching policies by conditioning generation on the overlapping actions from the previous chunk. PAINT Ho et al. (2026) instead selects the initial noise through flow inversion and repainting so that the generated chunk satisfies the executed-action prefix constraint. VLA-RAIL Zhao et al. (2025b) provides a model-agnostic asynchronous inference framework that temporally aligns newly arrived chunks, smooths intra-chunk trajectories, and blends successive chunks. RTR Wang et al. (2026) targets high-frequency latent-space policies and introduces a training-free Reuse-then-Refine strategy. Other methods Xue et al. (2025); Sendai et al. (2025); Tang et al. (2025) require additional training or policy modification. While these methods primarily address inference stalls and trajectory discontinuities introduced by asynchronous chunk switching, T^2Align focuses on preserving the temporal semantics of chunked policies at both the execution and chunk levels, providing plug-in support for off-the-shelf chunked policies in time-critical dynamic tasks.

5.3 Accelerating Imitation Learning Policy Inference

Existing efforts accelerate imitation learning policies by reducing redundant computation or optimizing runtime execution, including caching Xu et al. (2026); Wei et al. (2026); Ji et al. (2025), distillation Ye et al. (2025); Song et al. (2026), quantization Wang et al. (2025); Zhang et al. (2026), pruning Yang et al. (2026b;a); Liu et al. (2025), and runtime/kernel optimization Ma et al. (2025); Dai et al. (2025). These methods reduce the latency of producing each action chunk. However, for chunked policies, reaction latency is still limited by the remaining execution time of the current chunk. Based on asynchronous inference, T^2Align is complementary to these acceleration methods.

6 Conclusion

We present T^2Align, a robot manipulation system for deploying chunked imitation learning policies in time-critical dynamic manipulation tasks. T^2Align combines execution-level tempo alignment and chunk-level transition alignment. Execution-level tempo alignment preserves the policy-intended execution tempo under safety constraints, while chunk-level transition alignment enables smooth and consistent transitions between asynchronously returned chunks. Together, these mechanisms provide plug-in support for off-the-shelf chunked policies and improve their ability to react quickly and execute actions within narrow temporal windows.

Limitations and future work. Despite improving performance on time-critical dynamic tasks, T^2Align is currently evaluated mainly on manipulation tasks with robotic arms. To unlock broader application potential, we plan to validate T^2Align on humanoid robots performing simultaneous locomotion and manipulation tasks in future work.

7 AI USE STATEMENT

We used generative AI tools solely to assist with language polishing. All AI-assisted text was reviewed and revised by the authors, who take full responsibility for the final content.

8 REPRODUCIBILITY STATEMENT

We provide implementation details and experimental settings in the appendix. We will release the full source code of T^2Align and collected real-robot data for all four real-world tasks evaluated in this paper.

REFERENCES

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

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*, 2025.

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

Remi Cadene, Simon Aliberts, Francesco Capuano, Michel Aractingi, Adil Zouitine, Pepijn Kooijmans, Jade Choghari, Martino Russi, Caroline Pascal, Steven Palma, et al. Lerobot: An open-source library for end-to-end robot learning. *arXiv preprint arXiv:2602.22818*, 2026.

Chi-Lam Cheang, Guangzeng Chen, Ya Jing, Tao Kong, Hang Li, Yifeng Li, Yuxiao Liu, Hongtao Wu, Jiafeng Xu, Yichu Yang, et al. Gr-2: A generative video-language-action model with web-scale knowledge for robot manipulation. *arXiv preprint arXiv:2410.06158*, 2024.

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.

Yuntao Dai, Hang Gu, Teng Wang, Qianyu Cheng, Yifei Zheng, Zhiyong Qiu, Lei Gong, Wenqi Lou, and Xuehai Zhou. Actionflow: A pipelined action acceleration for vision language models on edge. *arXiv preprint arXiv:2512.20276*, 2025.

Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

Flexiv. Rizon. `https://www.flexiv.com/product/rizon`, 2026.

Trong-Bao Ho, Quang-Tan Nguyen, Thien-Loc Ha, Gia-Binh Nguyen, Viet-Thanh Nguyen, Long Dinh, Minh N Vu, Duy MH Nguyen, An Thai Le, and Ngo Anh Vien. Start right, arrive right: Asynchronous execution via initial noise selection. *arXiv preprint arXiv:2606.19774*, 2026.

Sigmund H Høeg, Yilun Du, and Olav Egeland. Streaming diffusion policy: Fast policy synthesis with variable noise diffusion models. *arXiv preprint arXiv:2406.04806*, 2024.

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.

Chia-Yu Hung, Qi Sun, Pengfei Hong, Amir Zadeh, Chuan Li, U Tan, Navonil Majumder, Soujanya Poria, et al. Nora: A small open-sourced generalist vision language action model for embodied tasks. *arXiv preprint arXiv:2504.19854*, 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.

Kangye Ji, Yuan Meng, Hanyun Cui, Ye Li, Shengjia Hua, Lei Chen, and Zhi Wang. Block-wise adaptive caching for accelerating diffusion policy. *arXiv preprint arXiv:2506.13456*, 2025.

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.

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

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

Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, et al. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024.

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. *arXiv preprint arXiv:2410.07864*, 2024.

Ziyan Liu, Yeqiu Chen, Hongyi Cai, Tao Lin, Shuo Yang, Zheng Liu, and Bo Zhao. Vla-pruner: Temporal-aware dual-level visual token pruning for efficient vision-language-action inference. *arXiv preprint arXiv:2511.16449*, 2025.

Yunchao Ma, Yizhuang Zhou, Yunhuan Yang, Tiancai Wang, and Haoqiang Fan. Running vlas at real-time speed. *arXiv preprint arXiv:2510.26742*, 2025.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Robotiq. Adaptive grippers. `https://robotiq.com/products/adaptive-grippers`, 2026.

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

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

Haoming Song, Delin Qu, Yuanqi Yao, Qizhi Chen, Qi Lv, Yiwen Tang, Modi Shi, Guanghui Ren, Maoqing Yao, Bin Zhao, et al. Hume: Introducing system-2 thinking in visual-language-action model. *arXiv preprint arXiv:2505.21432*, 2025.

Wenxuan Song, Jiayi Chen, Shuai Chen, Jingbo Wang, Pengxiang Ding, Han Zhao, Yikai Qin, Xinhu Zheng, Donglin Wang, Yan Wang, et al. Fast-dvla: Accelerating discrete diffusion vla to real-time performance. *arXiv preprint arXiv:2603.25661*, 2026.

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.

UFACTORY. UFACTORY X-ARM 7. `https://www.ufactory.us/product/ufactory-xarm-7`, 2026.

Hongyu Wang, Chuyan Xiong, Ruiping Wang, and Xilin Chen. Bitvla: 1-bit vision-language-action models for robotics manipulation. *arXiv preprint arXiv:2506.07530*, 2025.

Kunyun Wang, Yuhang Zheng, Yupeng Zheng, Jieru Zhao, and Wenchao Ding. Learning high-frequency continuous action chunks in latent space. *arXiv preprint arXiv:2605.24931*, 2026.

Yujie Wei, Jiahan Fan, Jiyu Guo, Ruichen Zhen, Rui Shao, Xiu Su, Zeke Xie, and Shuo Yang. Learning to accelerate vision-language-action models through adaptive visual token caching. *arXiv preprint arXiv:2602.00686*, 2026.

Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, et al. Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 2025.

Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. Vla-cache: Efficient vision-language-action manipulation via adaptive token caching. *Advances in Neural Information Processing Systems*, 38:164448–164473, 2026.

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

Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL `https://github.com/facebookresearch/hydra`.

Yantai Yang, Yuhao Wang, Zichen Wen, Luo Zhongwei, Chang Zou, Zhipeng Zhang, Chuan Wen, and Linfeng Zhang. Efficientvla: Training-free acceleration and compression for vision-language-action models. *Advances in Neural Information Processing Systems*, 38:40891–40914, 2026a.

Zebin Yang, Yijiahao Qi, Tong Xie, Bo Yu, Shaoshan Liu, and Meng Li. Dysl-vla: Efficient vision-language-action model inference via dynamic-static layer-skipping for robot manipulation. *arXiv preprint arXiv:2602.22896*, 2026b.

Wencheng Ye, Tianshi Wang, Lei Zhu, Fengling Li, Guoli Yang, and Hengtao Shen. Actdistill: General action-guided self-derived distillation for efficient vision-language-action models. *arXiv preprint arXiv:2511.18082*, 2025.

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.

Jingxuan Zhang, Yunta Hsieh, Zhongwei Wan, Haokun Lin, Xin Wang, Ziqi Wang, Yingtie Lei, and Mi Zhang. Quantvla: Scale-calibrated post-training quantization for vision-language-action models. *arXiv preprint arXiv:2602.20309*, 2026.

Xinyu Zhang, Yuhan Liu, Haonan Chang, Liam Schramm, and Abdeslam Boularias. Autoregressive action sequence learning for robotic manipulation. *IEEE Robotics and Automation Letters*, 2025.

Qingqing Zhao, Yao Lu, Moo Jin Kim, Zipeng Fu, Zhuoyang Zhang, Yecheng Wu, Zhaoshuo Li, Qianli Ma, Song Han, Chelsea Finn, et al. Cot-vla: Visual chain-of-thought reasoning for vision-language-action models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 1702–1713, 2025a.

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.

Yongsheng Zhao, Lei Zhao, Baoping Cheng, Gongxin Yao, Xuanzhang Wen, and Han Gao. Vla-rail: A real-time asynchronous inference linker for vla models and robots. *arXiv preprint arXiv:2512.24673*, 2025b.

Haoyu Zhen, Xiaowen Qiu, Peihao Chen, Jincheng Yang, Xin Yan, Yilun Du, Yining Hong, and Chuang Gan. 3d-vla: A 3d vision-language-action generative world model. *arXiv preprint arXiv:2403.09631*, 2024.

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

APPENDIX

A METHOD DETAILS

A.1 SYSTEM OVERVIEW OF T^{2}ALIGN

T^{2}Align is implemented as a modular execution system for off-the-shelf chunked imitation policies. It decouples three concerns that are often entangled in robot policy deployment: policy inference, chunk-level control, and physical execution. As shown in Fig. 7, the Inference Processor wraps the policy and serves inference requests; the Controller manages the chunk timeline and performs chunk-level transition alignment; and the Executor handles safety-constrained, tempo-aligned physical execution.

The system operates as follows. (1) The Observation thread runs in the background to collect the latest observations from the robot and (2) send them to the Controller, so that the Controller can use the most recent observation when launching inference. (3) The Controller requests asynchronous inference from the Inference Processor once the number of consumed actions in the current chunk exceeds the inference threshold T. (4) When a new asynchronous chunk returns, (5) the Controller first performs chunk-level transition alignment to align the returned chunk with the current execution timeline and stabilize how the new chunk takes over from the old one (Sec. 3.2). (6) The processed chunk is then inserted into a control queue and issued for execution in FIFO order. (7) During execution, the Executor performs execution-level tempo alignment online to compensate for tempo loss under safety constraints (Sec. 3.1). (8) After tempo alignment, the resulting actions are inserted into an execution queue, and (9) they are executed on the robot in FIFO order.

Benefiting from its decoupled design, T^{2}Align is easy to extend to off-the-shelf chunked imitation learning policies. Its hierarchical tempo and transition alignment enables smooth asynchronous chunk transitions to avoid instability, while keeping the execution tempo aligned with the intended tempo to avoid missing temporal windows. Therefore, from a systems perspective, T^{2}Align serves as a plug-in system enhancement that strengthens the reactive capability of chunked policies and enables strong performance and stability on time-critical dynamic tasks.

A.2 IMPLEMENTATION

T^{2}Align is implemented as a Hydra-configurable runtime for deploying off-the-shelf chunked imitation policies. We use Hydra Yadan (2019) to compose controllers, executors, environments, and policy wrappers from YAML configurations, enabling flexible system configuration. Components communicate through ZeroMQ request-reply channels: the Controller sends action and observation requests to the Executor and requests action chunks from a policy-serving process. T^{2}Align provides policy wrappers for Diffusion Policy Chi et al. (2025), ACT Zhao et al. (2023), and PI0.5 Intelligence et al. (2025). These wrappers normalize observations into each policy's expected format, run inference through PyTorch Paszke et al. (2019) or LeRobot Cadene et al. (2026) policy interfaces, and convert model outputs into executable position-control action targets.

B SUPPLEMENTARY EXPERIMENTS

B.1 EFFECT OF INFERENCE THRESHOLD ON REACTION AND STABILITY.

We evaluate the effect of the inference threshold on reaction and stability using the Whack-A-Mole task. Since we use a chunk size of 48 for this task, an inference threshold of 48 corresponds to synchronous inference. Lowering the inference threshold reduces reaction latency and therefore should improve responsiveness on time-critical dynamic tasks. As shown in Fig. 9, the experimental results confirm this trend. However, lowering the inference threshold also increases the frequency of chunk transitions, which leads

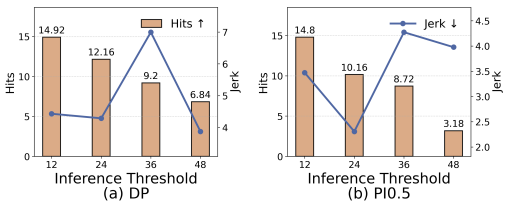

Figure 9: Effect of inference threshold on reaction and stability for Whack-A-Mole using DP and PI0.5.

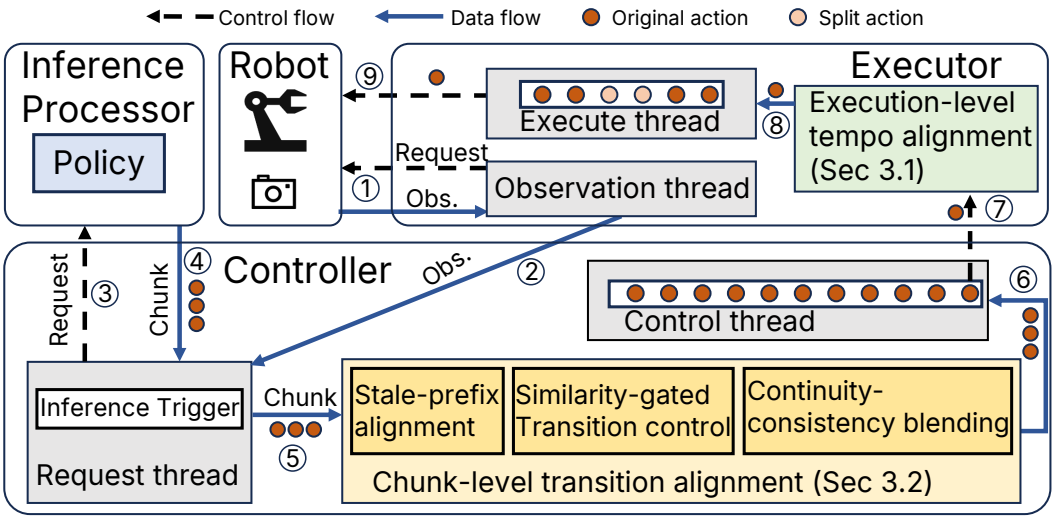

Figure 7: System overview of T^{2}Align.

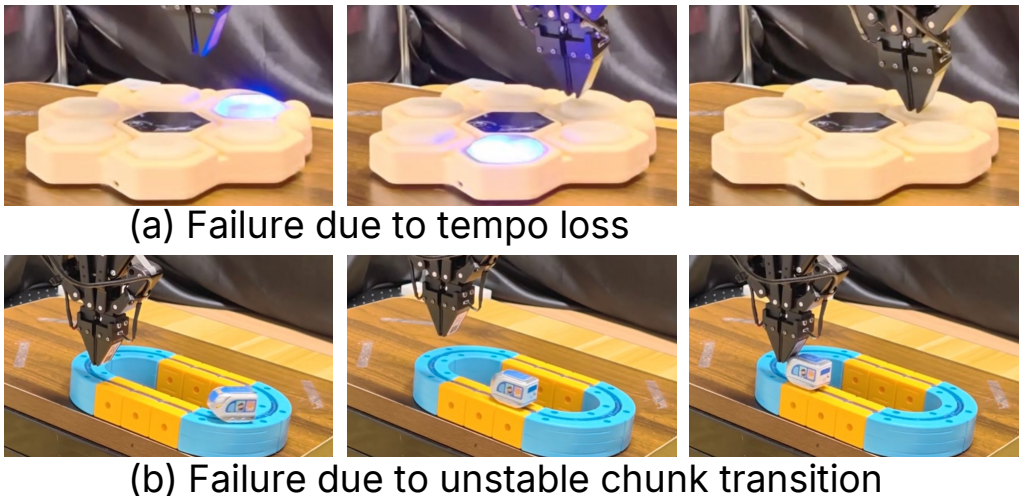

Figure 8: (a) Tempo loss causes the robot to miss the temporal window for hitting the illuminated button. (b) An unstable chunk transition causes tremor and leads to collision with the cart. Task definitions are given in Sec. 4.1.

to higher jerk at a threshold of 12 compared with 24. Further increasing the threshold to 36 causes the policy to miss the correct timing more often, resulting in both performance degradation and higher jerk. When the threshold is 48, corresponding to synchronous inference, the number of hits is the lowest. Meanwhile, because there are no asynchronous chunk transitions, the jerk is lower than that at a threshold of 36.

C Experimental Details

C.1 Hardware

Unless otherwise noted, all real-world evaluations are conducted on a 7-DoF UFACTORY xArm 7 robotic arm UFACTORY (2026) equipped with a Robotiq 2F-85 adaptive gripper Robotiq (2026). Data collection is performed on a local workstation with an NVIDIA RTX 3060 GPU. During evaluation, policy inference runs on a remote server with an NVIDIA RTX 4090 GPU, while real-time

Table 8: Training details for Diffusion Policy.

Hyperparameter	Value
Training Configuration	
World size (GPUs)	1
Batch size	64
Training epochs	450
Policy Configuration	
Noise scheduler	DDIM
Training diffusion steps	30
Inference diffusion steps	30
Optimizer & LR Schedule	
Optimizer	AdamW
Learning rate	1×10^{-4}
Weight decay	1×10^{-6}
Adam betas	$(0.95, 0.999)$
LR scheduler	Cosine decay
Warmup steps	500

robot control is performed over network communication. To evaluate cross-hardware applicability of T^{2}Align, we additionally conduct the Cart Gate task on a 7-DoF Flexiv Rizon 4s adaptive robot arm Flexiv (2026), using the same local-workstation and remote-inference setup. For simulator experiments, Timer Whack-a-Mole experiment in Sec. 4.3 is evaluated on Nvidia A100 GPU. And libero experiment is evaluated on Nvidia RTX 4090 GPU.

C.2 Experiment Setup Details

(1) **Whack-A-Mole**. The task uses a device with six illuminated buttons. At each target interval, one button is randomly activated. The robot must press the illuminated button before the target switches after a fixed interval of 2 seconds. Pressing the illuminated button is counted as a correct hit. (2) **Press Car**. The cart has a button on its top, and pressing the button changes the cart's moving direction. The robot must first press the button to activate the cart, then follow the moving cart and press the button again before the cart reaches a predefined failure region. A trial fails if the cart is not activated or enters the failure region. (3) **Cart Gate**. A cart moves along a circular track, and the robot arm initially blocks the cart's path. The robot must lift the arm before the cart arrives, allow the cart to pass, and lower the arm immediately afterward. A trial fails if the arm collides with the cart or remains raised after the cart passes. (4) **Press Timer**. We use Press Timer as a diagnostic task for execution-level tempo alignment. The robot presses the start button of a dual-button timer and then presses the stop button. The demonstrated interval between the two presses is constrained to the target duration range $[4.8, 5.2]$ seconds.

C.3 Training Details of Different Policies

We train the policies used in our experiments on NVIDIA A100 GPUs with 80 GB of memory. The training details for DP, ACT, and PI0.5 are shown in Table 8, Table 9, and Table 10, respectively.

C.4 Latency Breakdown for Different Policies

During deployment, the Inference Processor runs on a remote server, while the Controller and Executor run on an edge PC connected to the robot through a wired link. The Controller sends inference requests to the remote server over Wi-Fi. We report the latency breakdown in Table 11.

Table 9: Training details for ACT.

Hyperparameter	Value
Training Configuration	
World size (GPUs)	1
Batch size	64
Training steps	800,000
Policy Configuration	
Vision backbone	ResNet-18
Backbone weights	ImageNet pretrained
Transformer hidden dim	512
Attention heads	8
Encoder layers	4
Decoder layers	1
VAE latent dim	32
KL weight	10
Dropout	0.1
Optimizer & LR Schedule	
Optimizer	AdamW
Learning rate	1×10^{-5}
Weight decay	1×10^{-4}
Adam betas	$(0.9, 0.999)$
Gradient clipping norm	10
LR scheduler	None

Table 10: Training details for PI0.5.

Hyperparameter	Value
Training Configuration	
World size (GPUs)	8
Batch size	20
Training steps	80,000
Policy Configuration	
Precision	bfloat16
PaliGemma variant	gemma_2b
Action expert variant	gemma_300m
Optimizer & LR Schedule	
Optimizer	AdamW
Peak learning rate	7.5×10^{-5}
Final learning rate	7.5×10^{-6}
Weight decay	1×10^{-2}
Adam betas	$(0.9, 0.95)$
Gradient clipping norm	1
LR scheduler	Cosine decay with warmup
Warmup steps	1,000
Decay steps	80,000

D DEMO VIDEOS

We include the demonstration videos `demo_final.zip` in the Supplementary Material. The videos complement the quantitative results in the main paper by providing qualitative examples of T^2Align and the corresponding baselines on real robot platforms. The archive is organized by robot

Table 11: Latency breakdown under real-robot execution.

Component / Method	Mean (ms)	Std (ms)
System Overhead		
Network latency	45.55	2.51
Policy Inference Latency		
DP	122.48	12.29
ACT	7.07	0.70
PI0.5	135.27	6.48
Robot Execution Latency		
Robot execution latency	16.98	8.04

platform, task, experiment type, and policy. Videos labeled `baseline` correspond to the original policy deployment without T^{2}Align, while videos labeled `ours` correspond to deployment with T^{2}Align enabled. The labels `sync` and `async` denote synchronous and asynchronous inference, respectively.

Archive organization. The directory `xarm/` contains the main experiments on a UFACTORY xArm 7 robot arm. Under this directory, `whack_a_mole/`, `press_car/`, and `cart_gate/` contain end-to-end evaluation demo videos for DP, ACT, and PI0.5 on the three time-critical dynamic tasks. The directory `press_timer/` contains diagnostic demonstrations for PI0.5 that isolate execution-level tempo alignment. The directory `flexiv/` contains Cart Gate demonstrations on a Flexiv Rizon 4s robot arm, illustrating cross-hardware applicability.

End-to-end task executions. The end-to-end videos are located under `xarm/whack_a_mole/end2end/`, `xarm/press_car/end2end/`, and `xarm/cart_gate/end2end/`. Each task directory contains policy-specific subdirectories for DP, ACT, and PI0.5, and each policy directory compares synchronous and asynchronous baselines with their T^{2}Align counterparts. We recommend first watching `xarm/whack_a_mole/end2end/pi05/` for an intuitive overview, because Whack-A-Mole stresses fast reaction to changing targets and makes both responsiveness and transition stability visually apparent. The Press Car and Cart Gate videos provide additional qualitative examples on tasks whose success depends on completing actions within narrow temporal windows.

Component and sensitivity videos. The diagnostic videos in `xarm/press_timer/ablation/pi05/` compare PI0.5 with execution-level tempo alignment disabled and enabled. These videos provide qualitative examples of whether T^{2}Align preserves the policy-intended execution tempo and keeps the realized motion duration close to the intended temporal window.

The videos in `xarm/whack_a_mole/ablation_of_execute_level_tempo_align/` compare DP, ACT, and PI0.5 with execution-level tempo alignment disabled and enabled. They show how execution-level tempo alignment reduces the slowdown caused by safely executing large-stride actions, making the realized execution speed closer to the demonstrated tempo.

The videos in `xarm/whack_a_mole/ablation_of_continuity_consistency_transition_blending/` compare direct chunk switching, continuity-only blending, and continuity-consistency transition blending. These videos qualitatively illustrate that smoothing transitions based only on continuity can still produce unstable or inconsistent motions, whereas considering both continuity and consistency improves transition stability and task progress.

The videos in `xarm/whack_a_mole/ablation_of_similarity_gated_transition_control/` compare results with similarity-gated transition control disabled and enabled for DP and PI0.5. These demonstrations show that the gate improves stability by avoiding harmful blending when the queued and incoming chunks correspond to incompatible motion intents.

The videos in `xarm/whack_a_mole/ablation_inference_threshold/` sweep different asynchronous inference thresholds for DP and PI0.5. They provide qualitative examples of the trade-off between reaction latency and execution stability.

Cross-hardware demonstrations. The videos in `flexiv/flexiv_cart_gate/pi05/` show Cart Gate executions with PI0.5 on a Flexiv Rizon 4s robot arm. These videos compare synchronous and asynchronous baselines with T^2Align, and qualitatively support that the proposed tempo and transition alignment mechanisms are not specific to the xArm 7 platform.