

Who Needs Attention Anyway?

Retaining State Corrections in Recurrent Control

Dario Fumarola
Amazon Web Services
New York, New York

Abstract

A local optimizer can improve a recurrent controller’s action while leaving its memory unchanged. We study whether retaining the optimized state improves subsequent decisions, beyond the benefit of caching its action sequence. A frozen controller predicts a short plan, and bounded Gauss–Newton updates refine either the plan or the state that produces it. Paired simulator branches execute an identical present action before varying what is stored. Retention improves the next few proposals and reduces repeated optimization. In a matched plan-cache experiment, it also lowers eight-input realized cost by 0.7% in navigation and 1.1% in constrained chain control when both branches begin with the same corrected plan in their cache. In full navigation episodes, retaining rather than discarding state corrections raises success from 82.7% to 85.4%; warm-started model predictive control reaches 85.8% with substantially fewer contacts. Fitting an action-space solution into memory recovers most of the retention benefit. A simple rule rolls back the latest state correction when the target changes or sensing becomes stale, reducing the harm of retaining an outdated solution. These results identify recurrent-state correction as a short-lived complement to plan reuse.

1 Introduction

A local planner can correct a neural controller’s next action without changing the state that produced it. At the next observation, the recurrent model may propose much the same action again. This paper asks whether writing the correction into recurrent memory reduces that repeated work.

The distinction is easy to miss. Suppose a frozen recurrent model proposes a hidden state and decodes it into a short control sequence. Optimizing the hidden state changes the sequence. The controller can then execute the improved first action and either retain the optimized state or discard it. Both choices obtain the present action improvement. Only the first changes the memory passed to the next recurrent update.

A correction may encode a useful response to an obstacle that remains nearby, or favor a target that changes at the next input. Moreover, an action-space optimizer can retain its previous solution, warm-start the next solve, and often search more effectively than an optimizer restricted by a learned decoder. Model predictive control (MPC) is therefore a central comparison [1, 2]. The relevant question is whether correcting recurrent memory adds anything when a good plan is already retained.

We separate two questions. First, is state correction competitive with action refinement at a given latency? Second, holding the present action fixed, what changes when the corrected state is retained? The first requires strong control baselines. The second requires a direct intervention on memory. We execute an identical corrected action, then independently vary whether the controller retains the corrected state and whether it caches the corrected plan. This separates the current action improvement from the two ways of reusing its solution.

The correction rule uses a fresh, damped Gauss–Newton matrix, a limit on total state displacement, and a finite number of candidate evaluations. An accepted candidate must

decrease an objective computed from the current observations and a local dynamics model. The method changes neither the recurrent weights nor the decoder weights.

Navigation provides the main comparison with action refinement, predictive filtering, and warm-started MPC. A second task requires moving a redundant articulated chain to an end-effector target while respecting obstacles and joint-motion limits. The same branching experiment is used in both tasks, first without further optimization and then with a shared action-space solver.

The main finding is that a retained correction improves the next few recurrent proposals. Caching a plan captures part of this benefit, but state retention still helps when both branches begin with the same cached plan. Fitting an action-space solution back into the state obtains a similar improvement. A corrected state lets a frozen proposal generator carry a local solution into its next update.

2 Correcting a Recurrent Controller

2.1 A common objective in action coordinates

At input t , the controller receives observation x_t , including the action actually executed at $t - 1$. It proposes

$$z_t^0 = f_\theta(z_{t-1}, x_t), \quad U_t^0 = g_\phi(z_t^0; c_t), \quad (1)$$

where $z_t^0 \in \mathbb{R}^d$ is the complete persistent state, $U_t^0 = (u_{t,0}, \dots, u_{t,H-1})$ is an H -step control sequence, and c_t contains the current physical state estimate, target, and local geometry. The decoder includes projection onto $\mathcal{U}_t$, the task’s convex set of permitted control sequences. Only $u_{t,0}$ is executed.

Let $r_t(U; c_t)$ collect the weighted task residuals from a

rollout under the local model. We use

$$C_t(U) = \frac{1}{2} \|r_t(U; c_t)\|_2^2, \quad (2)$$

$$F_t(U) = C_t(U) + \frac{\mu}{2m} \|S(U - U_t^0)\|_2^2, \quad (3)$$

where m is the number of scalar controls and the diagonal matrix S divides each control by its physical limit. All quantities in F_t are held fixed during the solve, including the anchor U_t^0 .

Action refinement minimizes $F_t(U)$ directly. State correction minimizes

$$L_t(z) = F_t(g_\phi(z; c_t)). \quad (4)$$

Both use the same residuals, dynamics, physical limits, proximity weight, and cumulative action-change limit

$$\|S(U - U_t^0)\|_2 / \sqrt{m} \leq \rho_U. \quad (5)$$

There is no additional latent proximity penalty. State correction also imposes $\|z - z_t^0\|_2 \leq \rho_z$ to limit the change written into memory. This additional restriction can make action search more effective on the current objective; the experiments measure that difference explicitly.

The state coordinates are standardized once using training-set statistics. The recurrence and decoder incorporate the inverse transformation, preserving the trained controller's function. Projection, rollout, and residual evaluation are shared by all correction methods. Their derivatives use the active branch at a projection boundary.

2.2 A bounded local update

Write $L_t(z) = \|R_t(z)\|_2^2/2$, including the action-proximity residual. At the current accepted state z , form

$$b = J^\top R_t(z), \quad P = J^\top J + \lambda I, \quad v = -P^{-1}b, \quad (6)$$

where $J = \partial R_t(z)/\partial z$ and $\lambda > 0$. The solve is exact at the current linearization. It uses the decoder's output dimension rather than a dense $d \times d$ factorization; section A gives the formula.

Starting with $\alpha \leq 1$, choose a candidate $z' = z + \alpha v$ within the cumulative state ball. Accept it only if its decoded sequence satisfies eq. (5) and

$$L_t(z) - L_t(z') \geq -\eta \alpha b^\top v > \epsilon_t. \quad (7)$$

A rejected candidate consumes a trial. The next trial halves α and reuses the direction. After acceptance, a remaining trial may compute a new direction. Non-finite quantities end the solve. The reference configuration allows three trials per input.

This is a bounded version of damped Gauss–Newton with an Armijo decrease test [3]. The accepted loss decreases and the total state displacement is bounded. A short proof and the evaluation count are in section A.3.

Algorithm 1 One state-correction call

Require: z_t^0, c_t ; trial limit K ; radii ρ_z, ρ_U

- 1: $z \leftarrow z_t^0$; evaluate $L_t(z)$; mark direction stale
- 2: **for** $j = 1, \dots, K$ **do**
- 3: **if** direction is stale **then**
- 4: Form b, P, v by eq. (6); mark direction fresh
- 5: Stop if non-finite or $-\eta b^\top v \leq \epsilon_t$
- 6: Set largest $\alpha \in [0, 1]$ within the state ball
- 7: **end if**
- 8: $z' \leftarrow z + \alpha v$; decode and evaluate candidate
- 9: Stop if the candidate is non-finite or $\alpha = 0$
- 10: **if** candidate passes eqs. (5) and (7) **then**
- 11: $z \leftarrow z'$; mark direction stale
- 12: Stop if the accepted decrease is below tolerance
- 13: **else**
- 14: $\alpha \leftarrow \alpha/2$; retain the current direction
- 15: **end if**
- 16: **end for**
- 17: **return** $\tilde{z}_t = z$ and $\tilde{U}_t = g_\phi(z; c_t)$

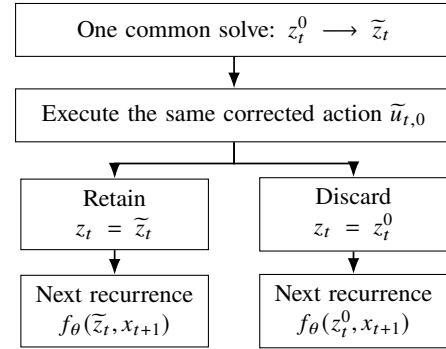


Figure 1. The memory intervention. Both branches begin with the same physical state and execute the same action. Their first subsequent observation is therefore also the same. Later actions can differ because only the stored recurrent state differs.

2.3 What is retained

The two state-correction controllers execute the same rule, differing only in the stored state:

$$\begin{aligned} \text{discard: } u_t &= \tilde{u}_{t,0}, \quad z_t = z_t^0, \\ \text{retain: } u_t &= \tilde{u}_{t,0}, \quad z_t = \tilde{z}_t. \end{aligned} \quad (8)$$

Their outputs are identical when started from a common proposal. On full episodes, their later proposals can differ because the recurrence receives different stored states. Both receive the action actually executed as an ordinary input, so the discard controller is not deprived of action feedback.

We also test *action refinement with state fitting*. First solve directly for U_t^* . Then fit a nearby state to that solution:

$$\min_{\|z - z_t^0\| \leq \rho_z} \frac{1}{2m} \|S(g_\phi(z; c_t) - U_t^*)\|_2^2. \quad (9)$$

One damped Gauss–Newton direction and at most two candidate evaluations are allowed. The first action of U_t^* is executed

regardless of the fitting result; the fitted state is used only at the next input. Fitting shares the total latency budget with action refinement. This comparison asks whether memory needs a task-gradient update, or whether it is enough to retain a good action-space solution.

2.4 Rolling back an outdated correction

The retain controller always passes $\tilde{z}_t$ to the next recurrence. A second variant keeps z_t^0 for one additional input, so that the latest correction can be undone before it is used. On receiving x_{t+1} , it sets

$$\hat{z}_t = z_t^0 + a_{t+1}(\tilde{z}_t - z_t^0), \quad a_{t+1} \in \{0, 1\}, \quad (10)$$

and computes $z_{t+1}^0 = f_\theta(\hat{z}_t, x_{t+1})$. The switch is zero if the active target identifier changes or the relevant sensor record is more than one input old; otherwise it is one. The action already executed at t is unchanged.

This rule uses target identifiers and sensor timestamps available to every controller. It adds one saved state vector and no learned parameters. It undoes only the latest correction, preserving the rest of the recurrent history. We call this variant *context rollback* and evaluate it separately from unconditional retention.

3 Related Work

MPC repeatedly optimizes a finite control horizon and applies its first input. Real-time iteration exploits successive problems' similarity through warm starts and bounded work per update [1]. Schwenkel et al. [2] go beyond shifting the last plan: they store earlier optimization data and use it to construct improved warm starts, with stability and asymptotic learning results under their assumptions. Differentiable MPC combines optimization with learned models and objectives [4].

Reusing an optimization result is therefore established practice. Our question concerns its location: outside the policy as a control sequence, or inside the policy as a changed recurrent state. The matched cache experiment compares these choices after a single common solve. Its one-plan cache is the temporal warm start, not the historical-data method of Schwenkel et al.

Safety filters modify a proposed control when it conflicts with constraints. Control-barrier-function filters express this intervention as a quadratic program [5]. Predictive safety filters use a model and a backup policy to reason over future constraint satisfaction [6]. Multi-step filtering can reduce repeated interventions and chattering without changing the nominal controller's memory [7]. This is an important alternative explanation for lower correction effort. We therefore include a multi-step predictive filter and let both action refinement and MPC retain their previous optimized control sequences.

Inference-time activation updates also precede this work. Plug and Play Language Models steer a frozen model by differentiating an auxiliary objective through its hidden activations [8]. Test-Time Training layers instead make a learned

model the recurrent hidden state and train the update procedure itself [9]. Here, the recurrent architecture is already trained and remains fixed. We compare the future consequences of retaining or discarding a bounded activation update.

Our comparison holds the present output fixed, changes the stored state, and follows the controller through either common-input replay or physical closed-loop continuation. This intervention measures what is gained by retaining an optimized activation.

4 Experimental Design

4.1 Tasks

Navigation. A radius-0.12 m agent moves through 10×10 m mazes with narrow doorways and six to twelve moving circles. It observes a local wall-distance grid, visible obstacle positions and estimated velocities, its velocity, the relative goal, and its previous action. A common waypoint process supplies local route information to every controller. The policy predicts 12 two-dimensional accelerations at 0.2 s intervals. The local objective penalizes waypoint error, small clearance, control effort, and abrupt control changes.

Success means reaching within 0.35 m of the goal in at most 160 inputs. Contacts are counted from swept physical motion, not the optimized clearance penalty. We report contact-containing steps per 10^3 executed steps and time to goal capped at 160 for failed episodes. A separate stress test delays obstacle observations by two inputs while retaining current agent proprioception.

Constrained chain control. A planar fixed-base chain has 12 links of length 0.18 m. Its input contains current joint angles, the previous executed joint velocities, four circular obstacles, and a two-dimensional end-effector target. It receives neither intermediate target coordinates nor a target joint configuration. The decoder predicts eight joint-velocity vectors at 0.1 s intervals.

Joint angles and velocities are bounded, and large velocity changes are penalized. Obstacles and nonadjacent links must be avoided throughout motion. Targets are generated from collision-free configurations, but those configurations are withheld from the controller. An endpoint may have many inverse-kinematic solutions; finding one does not specify a feasible transition from the current chain. The task therefore requires constrained motion rather than coordinate reconstruction.

An episode contains three targets, each held for 40 inputs. Success requires reaching within 0.04 m of every target for five consecutive inputs before it changes, with no contact during the episode. Endpoint error is averaged over the final five inputs of each target interval. Geometry, costs, and generation rules are specified in section B.

4.2 Controllers and compute matching

All learned-proposal methods share a frozen 256-state recurrent model and a two-layer decoder within each training seed. Action refinement keeps the best of the current neural proposal

and the projected, shifted solution from the previous input as its initial candidate. The extra shifted-plan check consumes a candidate slot; the neural-proposal evaluation is already required by both correction methods. The shifted solution is projected into the same action-change region around U_t^0 .

The MPC baseline uses the same horizon, observations, dynamics, and task costs. It removes the correction-specific proximity penalty and action-change radius, imposes predicted clearance constraints, and warm-starts both its control sequence and constraint multipliers. The current neural proposal is an alternative initial trajectory. Thus MPC can use its entire budget for control improvement rather than reproducing a bounded repair problem.

Navigation also includes a control-barrier-function (CBF) filter on the first action and a multi-step predictive filter. The latter minimizes deviation from the full nominal sequence subject to predicted clearance, actuator limits, and a terminal braking condition. These filters use the same observations and local model as the other controllers. Their formulations and infeasibility handling are given in section C.

Configurations are selected on validation episodes under two limits: 4 ms mean controller time and 8 ms p99. Solver iteration limits, damping, and stopping tolerances are tuned separately for each method. The retain/discard intervention is an exception: it shares every numerical setting, so the stored-state choice is the only change. The latency sweep uses additional envelopes, with the p99 target twice the mean target.

Neural inference and its derivatives run in FP32 on an NVIDIA A100-SXM4 80 GB. MPC and predictive filtering use compiled CPU solvers on the same workstation, an AMD EPYC 7543 with one solver thread. Timing includes controller-side transfers, warm-start checks, derivatives, failed candidates, state fitting, and output synchronization. Common sensor preprocessing and simulator advancement are excluded. This compares elapsed controller time on one CPU/GPU system; solver work counts are reported separately.

4.3 Sampling and paired comparisons

Navigation uses 40,000 training episodes and 4,000 validation episodes. Chain control uses 30,000 and 3,000. Five independently trained recurrent models are each tested on 2,000 navigation and 1,500 chain episodes. Methods within a seed share initial conditions and disturbance streams. No test episode is used for fitting or selection.

Intervals use 10,000 paired hierarchical bootstrap replicates, resampling trained models and then their episodes. Entries $a \pm b$ show the estimate and the larger rounded half-width of the 95% interval. Differences central to the argument are reported with their paired intervals and per-model effects in table 5. Timing uses five full-stream repetitions after 1,000 warm-up inputs; the reported value is the median repetition statistic.

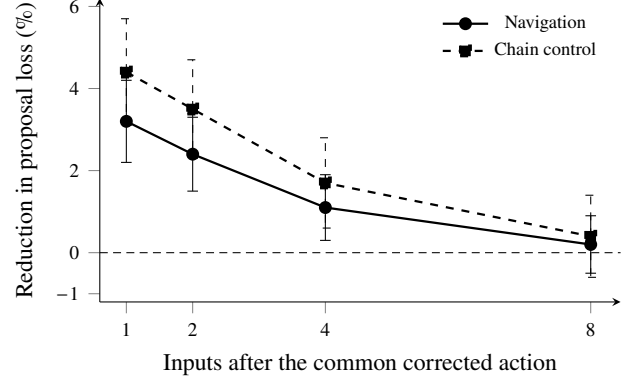


Figure 2. The direct memory effect decays over a few inputs. Both branches receive identical recorded observations, including previous-action features, and neither performs further correction. Error bars are paired 95% intervals.

5 Results

5.1 Identical present actions, different future decisions

We select 800 held-out episodes per trained model and one predetermined checkpoint in each. At the checkpoint, we clone the controller and simulator and run a single common state-correction call. Both branches execute its first action. One stores the corrected state and the other stores the proposal, as in fig. 1. Checkpoints with no accepted correction remain in the sample and have a zero initial intervention. There are 4,000 paired checkpoints per task, balanced across the five trained models.

The first continuation replays the same next eight observations and task contexts to both branches, with all correction disabled. It measures the effect on future proposals before changes in action can alter sensing. The task-loss reduction at horizon h is

$$B_h = 100 \frac{\bar{C}_h^{\text{discard}} - \bar{C}_h^{\text{retain}}}{\max\{\bar{C}_h^{\text{discard}}, 10^{-6}\}}, \quad (11)$$

where C_h excludes the proximity penalty and the bars average checkpoints before taking the ratio.

Retaining the state improves the next proposal by 3.2% in navigation and 4.4% in chain control (fig. 2). The improvement declines over four inputs and is close to zero by eight. The retained state is carrying a local adjustment, not a durable replacement for subsequent sensing or planning.

The second continuation advances two physical simulators. Each controller receives its own observations after actions diverge; exogenous disturbances are paired by absolute time. Correction is disabled for the next eight inputs. We evaluate the realized stage cost over those inputs, including a contact penalty and a terminal distance term. Retention lowers this closed-loop cost by 2.0% in navigation, [0.8, 3.2], and 3.0% in chain control, [1.4, 4.6].

A third continuation allows both branches to resume the same correction rule. Retention lowers realized cost by 1.0%

in navigation, [0.2, 1.8], and 1.6% in chain control, [0.4, 2.8], while also reducing optimization work. In navigation it uses 9.3 residual-Jacobian evaluations over the next eight inputs, compared with 12.1 after discarding the correction. In chain control the corresponding counts are 10.8 and 14.2. The paired reductions are 23.1%, [17.0, 29.0], and 23.9%, [17.5, 30.0]. The optimizer can repair a poor recurrent proposal again; retention reduces how often that repair must be rebuilt.

5.2 Does a cached plan explain the effect?

At the same checkpoints, we cross two choices: retain or discard $\tilde{z}_t$, and cache or discard $\tilde{U}_t$. All four branches execute the identical first action from the common solve. The plan-only branch keeps z_t^0 and the full corrected plan, giving it the optimization result without changing recurrent memory.

Over the next eight inputs, all branches use the same action-space Gauss–Newton solver. No further state correction or fitting is performed. The cache, where present, is shifted, projected into the current actuator set, and checked against the neural proposal; the better candidate initializes the solve. It is then replaced by the new optimized plan. Cache checks consume one of the three candidate slots and are included in timing.

For this comparison, the downstream solver minimizes C_t under actuator constraints, with no proposal-centered penalty or action-change radius. Retained state can change the initial proposal, but not the optimization objective or feasible set at a given physical context. All other settings are fixed across the four branches.

A plan cache alone lowers cost by 1.1% in navigation and 1.7% in chain control (table 1). When both branches begin with the same corrected plan in their cache, retaining the state lowers realized cost by a further 0.7% in navigation, [0.2, 1.2], and 1.1% in chain control, [0.3, 1.9]. These conditional effects are measured relative to the plan-only branch. They are positive in four of the five navigation models and all five chain models (table 5). The cached sequence does not fully replace the improvement to the recurrent proposal.

5.3 Action search is strong; retention adds a separate benefit

Retaining the corrected state raises navigation success from 82.7% to 85.4%, a paired gain of 2.7 percentage points with interval [1.7, 3.7] (table 2). The gain occurs without changing the current optimization rule. Contacts fall from 18.2 to 14.2 per thousand steps, and capped time falls by 5.6 inputs.

Direct action refinement is a stronger baseline than state correction with discarded memory: it reaches 83.9% success. On 4,000 common checkpoints, three candidate trials reduce the current objective by $30.8 \pm 1.0\%$ for action search and $27.6 \pm 0.9\%$ for state search. The checkpoint test starts both from the neural proposal, with no cached plan. Retain and discard are identical in this test. Action search is therefore not losing because it produces a worse immediate correction.

Fitting the action-space solution into memory raises success to 85.2%, a paired gain of 1.3 points, [0.5, 2.1], despite the

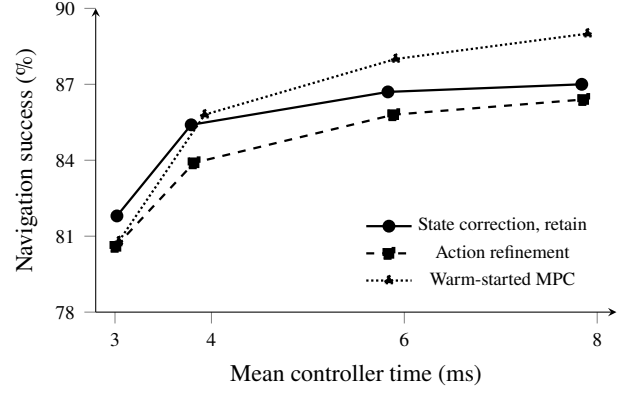


Figure 3. Navigation across latency envelopes. Each point is a separately validation-selected configuration. The state-retention advantage over action refinement is largest at the smaller budgets. MPC benefits more from additional solve time. The main operating points and uncertainty intervals are in table 2; lines connect the selected operating points.

fitting cost being charged to the same budget. Its difference from direct state retention is 0.2 points, [−0.5, 0.9]. These results favor retaining the optimization result over a particular choice of search coordinates.

MPC reaches 85.8% success, a difference of 0.4 points from retained state correction, [−0.4, 1.2]. The contact rates are much farther apart: 6.0 versus 14.2 per thousand steps. MPC reduces contacts by about 58% at nearly the same task-completion rate. The predictive filter reduces them further, to 4.1, but takes longer to reach goals. Retention improves the frozen controller; where contact avoidance dominates the objective, these results favor constrained planning or filtering.

The latency sweep in fig. 3 shows where this comparison changes. At about 3 ms, retained state correction reaches 81.8%, compared with 80.6% for action refinement and 80.8% for MPC. At about 8 ms, MPC reaches 89.0% and retained state correction reaches 87.0%. More optimization time favors direct planning; retention is most useful when each solve must remain small.

5.4 The effect also appears in constrained chain motion

On the chain, retention raises success from 74.2% to 78.6%, a paired gain of 4.4 points, [3.1, 5.7] (table 3). Fitting an action-space solution into memory again recovers most of that benefit. MPC has the highest success estimate and the lowest contact rate.

The same immediate-versus-future distinction holds. At common checkpoints, action search reduces the current objective by $34.9 \pm 1.2\%$, compared with $29.7 \pm 1.1\%$ for state search. Retention improves the repeated controller even though state search is less effective on a single frozen objective.

Table 1. Remembering a state, a plan, or both after the same action. All branches resume the same action-space solver. Cost reduction is relative to storing z_t^0 with no plan cache, over eight subsequent closed-loop inputs; positive is better. Times are per subsequent input and include cache handling. Only the initial recurrent-state correction is varied; a retained plan is refreshed after each solve.

Stored state and plan	Navigation			Chain control		
	Cost reduction (%)	Mean ms	p99 ms	Cost reduction (%)	Mean ms	p99 ms
z_t^0 , no plan cache	0	3.86	7.53	0	3.95	7.82
z_t^0 , cached plan	1.1 ± 0.7	3.64	7.25	1.7 ± 1.1	3.79	7.56
$\tilde{z}_t$, no plan cache	0.8 ± 0.7	3.72	7.39	1.4 ± 1.0	3.83	7.64
$\tilde{z}_t$, cached plan	1.8 ± 0.8	3.55	7.18	2.8 ± 1.2	3.70	7.43

Table 2. Navigation within the 4 ms mean, 8 ms p99 validation envelope. All methods have the same observations and 12-step model. Retain and discard share the state-correction solver and its settings. MPC and the filters have their own control objectives as described in the text. Lower contact rate, capped time, and latency are better.

Controller	Success (%)	Contacts/ 10^3	Capped time	Mean ms	p99 ms
Frozen recurrent policy	68.4 ± 1.9	37.1 ± 2.3	115.3 ± 2.8	2.10	2.70
CBF filter	79.2 ± 1.6	6.4 ± 0.8	100.5 ± 2.4	2.58	3.24
Multi-step predictive filter	82.1 ± 1.5	4.1 ± 0.6	95.0 ± 2.3	3.92	7.62
Action refinement	83.9 ± 1.4	15.9 ± 1.1	89.6 ± 2.2	3.82	7.31
Action refinement + state fitting	85.2 ± 1.3	14.4 ± 1.1	87.2 ± 2.1	3.91	7.64
State correction, discard	82.7 ± 1.5	18.2 ± 1.2	92.0 ± 2.3	3.81	7.43
State correction, retain	85.4 ± 1.3	14.2 ± 1.0	86.4 ± 2.1	3.79	7.28
Warm-started MPC	85.8 ± 1.3	6.0 ± 0.7	87.0 ± 2.1	3.93	7.71

5.5 Forgetting corrections when the context changes

A correction becomes less useful when the conditions that justified it change. Replacing the target immediately after the common action removes the average benefit of retention; on the chain, it increases subsequent cost (table 4, left). The new target is supplied normally to both branches.

Stale sensing causes a similar failure. With obstacle observations delayed by two inputs throughout navigation, success falls from 85.4% to 70.4% for retention and from 82.7% to 70.9% for discard. The fraction of evaluated candidates accepted changes only from 88% to 86%. The optimizer still finds improvements to the supplied objective, even when its geometry is out of date.

Context rollback (eq. (10)) withholds the latest correction when either change is visible in the input metadata. We test it on full episodes with fresh sensing interrupted by two eight-input bursts of two-input delay. Navigation bursts start at inputs 48 and 96; chain bursts start at 48 and 88. The targets and success criteria are unchanged. All three memory policies run the same state-correction solver at every input.

Rollback raises navigation success from 82.1% to 83.4%, a paired gain of 1.3 points, [0.5, 2.1]. The chain gain is 1.2 points, [0.4, 2.0] (table 4, right). Mean time increases by 0.02 ms or less, and both timing limits remain satisfied. On the nominal episodes, success is 85.3% in navigation and 79.0% on the chain, compared with 85.4% and 78.6% for unconditional retention. Rollback keeps nominal success close to unconditional retention while improving performance during delay bursts.

6 Discussion

The experiments separate three effects of a local solve: a better action now, a cached plan for the next solve, and a changed recurrent proposal. The identical-action intervention isolates the third. Its benefit lasts several inputs, appears in physical closed-loop continuation, and remains measurable when both controllers start with the same cached plan. Plan reuse accounts for part, but not all, of the effect.

This makes state correction a complement to an action-space optimizer. Direct action search is better at reducing the current objective in both tasks, and state fitting recovers most of the benefit of latent search. The practical choice is how to carry a useful solution forward. A plan cache can be inspected and re-evaluated explicitly. A retained state changes how the controller responds to the next observation, even before another solve. Using both gives the lowest mean cost in the matched experiment.

Retention also needs a way to forget. Keeping the uncorrected state for one input permits a cheap rollback when a command changes or a timestamp reveals stale sensing. This removes only the latest update, not older corrections already propagated through the recurrence. It also leaves the current action exposed to any error in the local model. Both tasks supply differentiable geometry and a target that can be scored before acting; the method depends on those inputs being useful.

The performance and safety conclusions differ. Retention improves success and contact rate relative to discard, but MPC has substantially fewer contacts at comparable success. State correction is not a substitute for constraint enforcement. The

Table 3. Constrained chain control. Success requires reaching all three end-effector targets without contact. Endpoint error is measured over the final five inputs of each target interval, including unsuccessful intervals. The latency envelope is the same as in navigation.

Controller	Success (%)	Endpoint error (cm)	Contacts/ 10^3	Mean ms	p99 ms
Frozen recurrent policy	58.6 ± 2.0	9.8 ± 0.5	24.6 ± 1.8	1.93	2.62
Action refinement	76.0 ± 1.6	5.1 ± 0.3	9.7 ± 1.0	3.86	7.51
Action refinement + state fitting	78.1 ± 1.5	4.5 ± 0.3	8.3 ± 0.9	3.96	7.83
State correction, discard	74.2 ± 1.7	5.6 ± 0.3	11.1 ± 1.1	3.91	7.73
State correction, retain	78.6 ± 1.5	4.4 ± 0.3	8.1 ± 0.9	3.88	7.58
Warm-started MPC	79.4 ± 1.5	4.1 ± 0.3	4.8 ± 0.7	3.97	7.85

Table 4. Changed context and selective rollback. Left: effect of one retained correction on eight-input physical cost, with further optimization disabled; positive means lower cost than discard. Right: full-episode success when sensing is intermittently delayed. Context rollback uses only the target identifier and record age, leaving the current action solver unchanged.

One-shot intervention: cost reduction (%)			Delay bursts: episode success (%)		
Continuation	Navigation	Chain	Memory policy	Navigation	Chain
Current sensing, fixed target	2.0 ± 1.2	3.0 ± 1.6	Discard	81.9 ± 1.5	70.9 ± 1.8
Target replaced at next input	-0.8 ± 0.9	-1.9 ± 1.1	Always retain	82.1 ± 1.5	71.8 ± 1.7
Records delayed by two inputs	-0.5 ± 1.0	-1.4 ± 1.2	Context rollback	83.4 ± 1.4	73.0 ± 1.7

evidence also comes from one gated recurrent architecture in two simulators. Its memory timescale, and the value of rollback, may change with the backbone or with training that includes state corrections.

7 Conclusion

An inference-time solve can improve more than the action it produces. Retaining its state correction improves the next few recurrent proposals and reduces repeated optimization, including when the corrected action sequence is already cached. The extra benefit is modest and short-lived. Direct action refinement with state fitting offers another way to obtain it, while context rollback helps discard corrections that have outlived their use. Remembering a plan improves the next solve; retaining its state correction can also improve the proposal that starts that solve.

References

- [1] M. Diehl, H. G. Bock, and J. P. Schlöder. A real-time iteration scheme for nonlinear optimization in optimal feedback control. *SIAM Journal on Control and Optimization*, 43(5):1714–1736, 2005. doi:10.1137/S0363012902400713.
- [2] L. Schwenkel, M. Gharbi, S. Trimpe, and C. Ebenbauer. Online learning with stability guarantees: A memory-based warm starting for real-time MPC. *Automatica*, 122:109247, 2020. doi:10.1016/j.automatica.2020.109247.
- [3] J. Nocedal and S. J. Wright. *Numerical Optimization*. Second edition. Springer, 2006. doi:10.1007/978-0-387-40065-5.
- [4] B. Amos, I. D. Jimenez Rodriguez, J. Sacks, B. Boots, and J. Z. Kolter. Differentiable MPC for end-to-end planning and control. *Advances in Neural Information Processing Systems*, 31, 2018. arXiv:1810.13400.
- [5] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control*, 62(8):3861–3876, 2017. doi:10.1109/TAC.2016.2638961.
- [6] K. P. Wabersich and M. N. Zeilinger. A predictive safety filter for learning-based control of constrained nonlinear dynamical systems. *Automatica*, 129:109597, 2021. doi:10.1016/j.automatica.2021.109597.
- [7] F. Pizarro Bejarano, L. Brunke, and A. P. Schoellig. Multi-step model predictive safety filters: Reducing chattering by increasing the prediction horizon. In *IEEE Conference on Decision and Control*, 2023. arXiv:2309.11453.
- [8] S. Dathathri, A. Madotto, J. Lan, J. Hung, E. Frank, P. Molino, J. Yosinski, and R. Liu. Plug and play language models: A simple approach to controlled text generation. In *International Conference on Learning Representations*, 2020. arXiv:1912.02164.
- [9] Y. Sun, X. Li, K. Dalal, J. Xu, A. Vikram, G. Zhang, Y. Dubois, X. Chen, X. Wang, S. Koyejo, T. Hashimoto, and C. Guestrin. Learning to (learn at test time): RNNs with expressive hidden states. In *International Conference on Machine Learning*, PMLR 267:57503–57522, 2025. PMLR 267, sun25h.
- [10] R. Verschuere, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl. acados: A modular open-source framework for fast embedded optimal control. *Mathematical Programming Computation*, 14:147–183, 2022. doi:10.1007/s12532-021-00208-8.

A Solver Details and Local Properties

A.1 Exact Gauss–Newton through the output space

At a fixed input, let $D = \partial g_\phi(z; c_t)/\partial z$ and let Q be the residual Jacobian with respect to the physical control sequence. Thus $J = QD$. The action-proximity term is part of Q , not a separate latent regularizer. Set

$$A = (Q^\top Q)^{1/2} D \in \mathbb{R}^{m \times d}. \quad (12)$$

Then $P = \lambda I + A^\top A$, and the inverse-vector product is

$$P^{-1}b = \lambda^{-1} [b - A^\top (\lambda I + AA^\top)^{-1} Ab]. \quad (13)$$

The square root uses a symmetric eigendecomposition, with negative round-off eigenvalues clipped to zero. Cholesky solves the inner system. Here $d = 256$ and $m = 24$ or 96 . This is an exact current-metric factorization, with all derivative and factorization work included in timing.

Action refinement solves its constrained quadratic in physical coordinates. Line-search candidates are projected into the actuator set and action-change ball; the decrease prediction uses the actual projected displacement.

A.2 Stopping and cumulative limits

The reference state solver uses $K = 3$, $\lambda = 0.02$, $\rho_z = 0.35$, $\rho_U = 0.20$, $\mu = 0.05$, and $\eta = 0.1$. The threshold is $\epsilon_t = 10^{-6} \max\{1, L_t(z_t^0)\}$. A further direction is not attempted after an accepted relative improvement below 10^{-3} of $\max\{1, L_t(z_t^0)\}$.

For displacement $e = z - z_t^0$ and a nonzero direction v , the largest nonnegative step within the state ball is the positive root of

$$(v^\top v)\alpha^2 + 2(e^\top v)\alpha + e^\top e - \rho_z^2 = 0, \quad (14)$$

clipped to $[0, 1]$. The action cap is checked on the decoded candidate, not on its linearization. A violation consumes a trial and halves the step. At zero step length the solve stops. A non-finite candidate stops rather than being retried. Rejected candidates never overwrite an accepted state.

State fitting uses the same state ball and damping. A candidate must reduce the fitting loss; failed fitting still counts toward the budget.

A.3 What the update guarantees

Proposition 1. *Suppose objective evaluations are deterministic and finite. Let $\mathcal{A}$ index the accepted steps of Algorithm 1, with directions v_j and step lengths α_j . Then*

$$L_t(\tilde{z}_t) \leq L_t(z_t^0) - \sum_{j \in \mathcal{A}} \eta \alpha_j b_j^\top P_j^{-1} b_j, \quad (15)$$

$$\|\tilde{z}_t - z_t^0\|_2 \leq \rho_z. \quad (16)$$

The call performs at most K candidate evaluations and K residual-Jacobian evaluations, in addition to its initial objective evaluation.

Proof. Positive damping gives $b_j^\top v_j = -b_j^\top P_j^{-1} b_j \leq 0$. Summing the accepted decrease tests proves the first statement. Every accepted state is checked against the same ball around z_t^0 , proving the second. Each loop slot evaluates at most one candidate and forms at most one Jacobian; a retry reuses the previous Jacobian. Early stopping can only reduce these counts. $\square$

This argument uses evaluated losses, including at projection boundaries. The implementation rejects non-finite quantities and uses the stated threshold against round-off. The result concerns one fixed local objective.

B Task Specification

B.1 Navigation

A 10×10 m region is split recursively into three to six rectangular rooms, each at least 2 m wide. Split positions are uniform between 40% and 60% of the parent’s longer side. Walls are 0.15 m thick; openings are uniform between 0.65 and 1.05 m. Start and goal lie in different connected rooms at least 5 m apart.

The common route process computes a shortest path through the known static room graph, ignoring moving obstacles. Its waypoint is the next doorway center, or the goal in the final room. This process is identical for the neural policy, action refinement, filters, and MPC. No method receives future moving-obstacle states. This keeps the comparison about local control rather than access to a global route.

Circle radii are uniform in $[0.12, 0.25]$ m. Each circle remains in its assigned room. With $\Delta t = 0.2$ s, its velocity evolves as

$$w_{i,t+1} = \text{proj}_{\|w\| \leq 0.25} (0.95 w_{i,t} + 0.08 \sqrt{\Delta t} \xi_{i,t}), \quad (17)$$

where $\xi_{i,t} \sim \mathcal{N}(0, I_2)$. Positions are advanced with this velocity and reflected at room boundaries. Reflection and future innovations are not included in the controller’s constant-velocity obstacle prediction.

The observation includes a 17×17 static wall-distance grid over a 6 m square, up to twelve circle records within 3 m, the relative waypoint and goal, current velocity, and the previous action. Each circle record contains relative position, velocity, radius, and a validity mask. Velocity estimates have independent noise of standard deviation 0.02 m/s per component. Unobserved slots contribute zero residual.

Agent dynamics are

$$v_{h+1} = (1 - \gamma \Delta t) v_h + \Delta t u_h, \quad p_{h+1} = p_h + \Delta t v_{h+1}. \quad (18)$$

The control limits are $|u_h| \leq 1$ m/s² componentwise, and $\gamma = 5/6$ s⁻¹. This drag keeps each velocity component within 1.2 m/s from an initially resting state. Control changes are penalized relative to the actual previous action u_{-1} .

For $H = 12$, define $b_\sigma(a) = \sigma \log(1 + e^{a/\sigma})$ and clearance margin $a_c = 0.15$ m. Let d_{hj} be predicted clearance from the agent’s boundary to object slot j . There are fourteen slots:

twelve circles, walls, and the boundary of the sensed square. With fixed padded normalization,

$$C_t^{\text{nav}}(U) = \frac{1}{2H} \sum_{h=0}^{H-1} \|(p_{h+1} - p_t^{\text{wp}})/3\|_2^2 + \frac{10}{28H} \sum_{h=0}^{H-1} \sum_{j=1}^{14} \left[\frac{b_{0.03}(a_c - d_{hj})}{a_c} \right]^2 + \frac{0.05}{2H} \sum_h \|u_h\|_2^2 + \frac{0.10}{2H} \sum_h \|u_h - u_{h-1}\|_2^2. \quad (19)$$

The static grid is bilinearly interpolated. The physical simulator checks swept contact; contact removes motion into the obstacle and retains the last nonpenetrating position. It does not terminate a navigation episode. The next controller input uses the resulting measured velocity and executed action.

B.2 Constrained chain

Let $q \in \mathbb{R}^{12}$ contain joint angles. With $X_0 = (0, 0)$ and $\ell = 0.18$ m,

$$X_j(q) = \ell \sum_{i=1}^j \begin{bmatrix} \cos(\sum_{k=1}^i q_k) \\ \sin(\sum_{k=1}^i q_k) \end{bmatrix}, \quad j = 1, \dots, 12. \quad (20)$$

Controls are joint velocities: $q_{h+1} = q_h + 0.1u_h$. The convex actuator set imposes $|q_h| \leq 2.6$ rad and $|u_h| \leq 1.2$ rad/s over the horizon. Velocity changes are penalized using 0.3 rad/s as their scale. Links have radius 0.01 m. Contact occurs when a link capsule intersects an obstacle or a nonadjacent link. In delayed-sensing tests, a commanded motion that would cross a physical joint limit is stopped at that limit and counted as a constraint contact.

Four static obstacle centers are sampled in the reachable disk with radii in $[0.08, 0.16]$ m. Initial configurations and three target configurations are sampled from smooth angle sequences in $[-1.8, 1.8]^{12}$ and rejected if they collide. Only each target's endpoint is supplied. Targets must be separated by at least 0.3 m. An offline planner screens for a collision-free, actuator-feasible transition reaching each target within 40 inputs; its intermediate configurations are not provided at inference. This screening is applied to the task generator before splitting data, not to individual controller outcomes.

For $H = 8$, let $e_h = (X_{12}(q_{h+1}) - p_t^*)/0.5$, and let d_{hj} range over 48 link-obstacle and 55 nonadjacent link-pair clearances. The task cost is

$$C_t^{\text{chain}}(U) = \frac{1}{2H} \sum_h \|e_h\|_2^2 + \frac{1}{2} \|e_{H-1}\|_2^2 + \frac{8}{206H} \sum_h \sum_{j=1}^{103} \left[\frac{b_{0.01}(0.03 - d_{hj})}{0.03} \right]^2 + \frac{0.02}{24H} \sum_h \|u_h/1.2\|_2^2 + \frac{0.05}{24H} \sum_h \|(u_h - u_{h-1})/0.3\|_2^2. \quad (21)$$

The minimum segment distance is differentiated on its active closest-point branch. Swept contact checking subdivides each input interval into twenty joint-space interpolation steps; the task's reported contact definition is this fixed-resolution check. On contact, motion stops at the last clear substep and that episode is marked unsuccessful. Evaluation continues to collect endpoint error and contact counts.

C Baselines, Training, and Evaluation

C.1 Control baselines

Action refinement retains its previous optimized plan, not a corrected recurrent state. Retain and discard instead start from the current recurrent proposal, with no separate plan cache. The immediate-loss checkpoint test removes all warm starts. The separate four-branch experiment in section 5.2 varies the plan cache explicitly.

MPC minimizes C_t subject to the actuator limits and predicted clearance $d_{hj} \geq a_c$ in navigation or $d_{hj} \geq 0.03$ m for the chain. It uses acados [10] with Gauss-Newton cost Hessians and warm-started primal and dual variables. The reference envelope permits three sequential quadratic programming (SQP) iterations and up to three line-search evaluations per iteration. Both the neural and shifted plans are checked before solving.

Predictive filtering uses the same SQP implementation and clearance constraints, with objective

$$\frac{1}{2H} \sum_{h=0}^{H-1} \|u_h - u_{t,h}^0\|_2^2 + \frac{0.05}{2H} \sum_h \|u_h - u_{h-1}\|_2^2, \quad (22)$$

plus terminal $\|v_H\|_\infty \leq 0.03$ m/s. Penalizing deviations over the full sequence follows the multi-step filtering rationale of [7]. This finite-horizon implementation uses estimated obstacle motion, a braking endpoint, and slack restoration. It is not the robust backup-set construction of [6].

For the CBF filter, let r and v_r be the relative position and velocity to a visible circle, and let $h = \|r\|_2^2 - R^2$, with R the summed radii plus the margin. The QP minimizes squared deviation from the nominal first acceleration subject to actuator limits and

$$2\|v_r\|_2^2 + 2r^\top(u - \gamma v_t) + 4\omega r^\top v_r + \omega^2 h \geq -\xi, \quad (23)$$

using constant obstacle velocity, $\omega = 2 \text{ s}^{-1}$, and $\xi \geq 0$. Walls use the corresponding linear-distance constraint. The slack penalty is $10^4 \|\xi\|_1$. No unobserved obstacles enter the QP.

MPC and predictive filtering use the same 10^4 penalty when predicted clearance or terminal constraints need slack. Actuator constraints are never relaxed. If a solve fails numerically, the controller checks its shifted plan and a maximum admissible braking plan, then executes the lower-slack candidate. Initialization, restoration, and fallback evaluation count toward time and evaluation limits. Solver slacks are not interpreted as physical safety guarantees.

C.2 Backbone and training

The backbone retains four 64-dimensional blocks, concatenated as a 256-dimensional state. A width-256 feature stream passes through four gated feedforward blocks of width 512. At block b ,

$$\tilde{a}_b = a_b + V_b [\text{SiLU}(W_b \text{LN}(a_b)) \odot W'_b \text{LN}(a_b)], \quad (24)$$

$$\gamma_b = 0.05 + 0.94 \text{sigmoid}(A_b \tilde{a}_b + \alpha_b), \quad (25)$$

$$h_{b,t} = \gamma_b \odot h_{b,t-1} + (1 - \gamma_b) \odot \tanh(B_b \tilde{a}_b + \beta_b). \quad (26)$$

For $b < 4$, set $a_{b+1} = \tilde{a}_b + C_b h_{b,t}$. The first feature vector is a learned projection of x_t . The decoder is a $256 \rightarrow 512 \rightarrow m$ multilayer perceptron with a SiLU hidden activation, followed by the actuator projection. All recurrent blocks, not just the final readout, are included in the corrected state.

The teacher uses the student’s sensed context, 30 SQP iterations, and eight initial control sequences. Training minimizes squared error on projected actions with AdamW: learning rate 3×10^{-4} , cosine decay, weight decay 10^{-4} , clipping at one, and 64 subsequences of length 64 per batch. Each subsequence has a 32-input burn-in. Navigation uses 100,000 updates and the chain 80,000. Checkpoint selection maximizes validation success, breaking ties by contact rate.

Training-rollout means and standard deviations define $z = D^{-1}(h - \bar{h})$, with standard deviations floored at 10^{-2} . Recurrence and decoder incorporate the inverse transform. Weights and scaling statistics are frozen before evaluation.

C.3 Selection and timing

Validation searches damping values 0.005, 0.02, and 0.08; candidate limits of one, two, three, five, or eight; and relative stopping thresholds of 10^{-4} , 10^{-3} , or 10^{-2} . All correction methods use $\mu = 0.05$ and $\rho_U = 0.20$. State radii are selected from 0.20, 0.35, and 0.50; both tasks select 0.35. Retain and discard share the retain-selected settings. MPC and predictive filtering search the same five choices of SQP iteration limit. Each QP has at most twenty interior-point iterations. Every configuration uses the same validation episodes.

Selection maximizes validation success within both latency targets, then minimizes contact rate and mean time. Settings are fixed before test evaluation. Retain and discard share three trial slots. Action refinement with fitting also has three candidate slots at the reference envelope: at most two for action search, including the shifted-plan check, and one for fitting.

Neural inference uses PyTorch 2.2 and CUDA 12.1; SQP models and derivatives are compiled before timing. The controller process is pinned to one physical CPU core without concurrent jobs. Timing runs from preprocessed host observation to completed host control, including transfers and fallback work. Five complete-stream sweeps follow warm-up, with independently permuted episode order. Their mean times vary by at most 0.09 ms and p99 by at most 0.23 ms. A p99 is a measured quantile, not a deadline.

C.4 Branch interventions and physical cost

One checkpoint time is chosen before each donor episode: inputs 20 through 120 in navigation, or 8 through 24 of the chain’s second target interval. Navigation donors continue for 160 inputs, holding the goal after arrival, so sampling does not depend on success. Half the donors use action refinement and half use state correction with discard. Each checkpoint saves the plant, controller, two-input observation history, and exogenous continuation draws.

The correction and first action are computed once and copied. Empty corrections remain as zero interventions; 61% of navigation and 68% of chain checkpoints have an accepted update. Replay observations come from a separately continued discard branch with correction disabled, including its previous-action features. The active waypoint or endpoint is held fixed for eight inputs, except in the target-replacement test.

Physical branches generate their own observations. Noise and obstacle innovations are indexed by absolute input and sensor or object slot, pairing exogenous draws without forcing equal observations after actions diverge. Every checkpoint enters the comparison. Before the common solve, inherited optimizer workspaces and plan caches are cleared in every branch.

For navigation, the realized cost over $n = 8$ future inputs is

$$J_n^{\text{nav}} = \frac{1}{n} \sum_{h=1}^n [\| (p_{t+h} - p_{t+h}^{\text{wp}}) / 3 \|_2^2 + 0.05 \| u_{t+h} \|_2^2 + 10 \mathbf{1}\{\text{contact at } t+h\} + \| (p_{t+n+1} - p_{t+n+1}^{\text{wp}}) / 3 \|_2^2]. \quad (27)$$

The indexing starts with the first decision after the common action; p_{t+h} is the measured state before that decision. The chain uses endpoint error normalized by 0.5 m, joint-velocity cost $0.02 \| u / 1.2 \|_2^2 / 12$, the same contact penalty, and a terminal endpoint term. Reported percentage reductions use ratios of mean physical costs, as in eq. (11). Active-correction continuations use the same stopping threshold in both branches; residual-Jacobian counts include evaluations followed by rejection or immediate termination.

For target replacement, the next navigation target is a different reachable waypoint in the current room, and the next chain target is a feasible endpoint at least 0.3 m from the old target, screened by the same generator. The new reference is announced at the next input. For delayed sensing, navigation delays obstacle records by two inputs but retains current proprioception. Chain control delays joint-angle measurements by two inputs while retaining the previous executed command; neither controller extrapolates the delayed quantities. These tests change the supplied context, not the accepted correction rule.

C.5 Plan-cache continuation

The four branches in table 1 start from the same checkpoint and common solve as the other interventions. The cached sequence is exactly $\tilde{U}_t$, including at checkpoints without an

Table 5. Paired effects for each independently trained model. The first pair of columns is the full-episode retain-minus-discard success difference. The middle pair is the reduction in eight-input physical cost from retaining state when both branches cache a plan. The last pair is the success gain from context rollback over unconditional retention during delay bursts. Positive values favor retention or rollback, respectively. Success differences are in percentage points (pp); cost differences are relative percentages.

Model	Retain vs. discard (pp)		State given plan cache (%)		Rollback vs. retain (pp)	
	Navigation	Chain	Navigation	Chain	Navigation	Chain
1	1.40	2.60	-0.1	0.2	0.30	0.40
2	2.00	3.40	0.4	0.6	0.80	0.73
3	2.50	4.20	0.6	1.0	1.20	1.07
4	3.35	5.60	1.1	1.5	1.80	1.60
5	4.25	6.20	1.5	2.2	2.40	2.20

accepted state update. Each cached plan is shifted by one input and padded by repeating its last control, then projected onto the current actuator set. The action-space solver evaluates the projected plan against the fresh neural proposal and chooses the lower task cost, breaking ties in favor of the neural proposal. A cache check uses a trial even when the plan is rejected.

All four branches have three candidate slots, damping 0.02, Armijo coefficient 0.1, and the same relative stopping threshold. Only the task residuals enter this diagnostic solve: $\mu = 0$, with no proposal-relative radius. A direction is accepted using its actual projected displacement. Cache maintenance and every candidate check count toward latency. The initial state intervention occurs once; subsequent recurrence is uncorrected in all four branches, while a plan cache is refreshed wherever that factor is enabled.

Cost reductions are ratios of mean physical costs across checkpoints. To measure the state contribution given a cache, we compare the state-plus-plan branch directly with the plan-only branch. This conditional ratio differs slightly from subtracting their reductions against the no-cache reference. Per-model effects use the same ratio within each model’s 800 checkpoints.

C.6 Context rollback and per-model effects

An identifier is assigned to each active navigation waypoint and each commanded chain endpoint. Identifiers change on ordinary target transitions as well as on replacement commands. Navigation uses the obstacle-record timestamp; chain control uses the joint-angle timestamp. Target identifiers and timestamps are available to every method but are not appended to the frozen network’s features. The one-input age threshold is fixed before evaluation and uses no simulator truth.

Rollback is applied before the next recurrent update. After that update, the old proposal/correction pair is released and replaced by the new pair. Once a previous correction has propagated into later states, this rule cannot reconstruct a never-corrected trajectory. In the delay-burst experiment, all methods receive the same delayed records, targets, and disturbance streams; sensing outside the two specified bursts is current. The nominal test retains the ordinary target transitions. No solver settings are retuned for rollback or delayed sensing. On delay-burst episodes, rollback takes 3.87/7.50 ms (mean/p99) in navigation and 3.94/7.74 ms in chain control.