TRACE: Trajectory-Routed Causal Memory for Delayed-Evidence Visuomotor Imitation

Anonymous Author(s)
Affiliation
Address
email

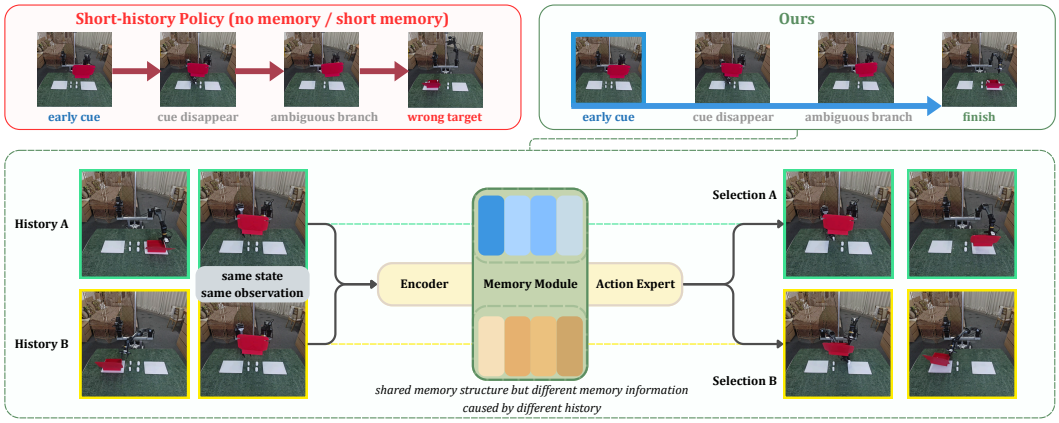

Figure 1: **Delayed evidence in long-horizon manipulation.** At the branch point, when the robot must choose one task continuation, the early cue is no longer visible, and current observations can look similar even though they require different actions. A short-history policy fails because its window contains the latest information but not the early cue. TRACE stores the cue when it is visible and reads that memory later to choose the branch.

Abstract: Long-horizon robot manipulation can require a robot to choose between task alternatives using evidence that is no longer visible. We study *delayed-evidence* tasks in which an early cue disappears before the later decision point, so similar current observations can require different actions. The current observation is therefore not a sufficient state for control, and the policy needs a compact record of earlier observations and actions to identify the correct branch. We introduce TRACE, a trajectory-routed causal evidence memory for visuomotor imitation policies. TRACE stores task-relevant visual evidence in a fixed-size latent memory and updates it online as the robot acts. Rather than organizing memory by raw time or manually provided task labels, TRACE uses *path signatures*, which provide fixed-size summaries of the executed trajectory, to organize evidence according to how the robot reached the current state. When the robot encounters an ambiguous observation, the policy conditions on this memory to recover the missing context and select the correct action. TRACE attaches through lightweight adapters, allowing it to condition regression-style action-chunking and diffusion policies without changing the policy backbone, action head, or imitation learning objective. We demonstrate TRACE on real-world manipulation tasks, showing that it improves policy performance when early evidence must be retained for later decisions.

1 Introduction

Robots often need to choose a later task branch using information that has already left view. A branch is a choice among task alternatives, such as which target, route, or manipulation routine to execute next. In delayed-evidence manipulation, an early cue is no longer visible at the branch point, where current observations from different histories can look nearly identical but call for different actions. Most visuomotor policies condition on the current observation, possibly with a short recent window, and therefore assume that this input is sufficient for action selection. This assump-

Submitted to the 10th Conference on Robot Learning (CoRL 2026). Do not distribute.

tion breaks in long-horizon manipulation because the present observation does not contain enough information to determine the correct branch. It violates the *Markov assumption* for control.

Adding more history is an incomplete solution. Short windows fail once the decisive evidence falls outside the window, while long windows increase cost and force the policy to infer which early observations still matter. Recurrent policies and generic memories can carry information forward, but cues needed for the later choice may be overwritten or mixed with task-progress signals that are irrelevant to that choice. Such *delayed-evidence* tasks require a causal, fixed-budget memory that preserves early task evidence and makes it available when the current observation becomes ambiguous.

We introduce TRACE, a trajectory-routed causal evidence memory for visuomotor policies when current observations are incomplete. TRACE stores task-relevant visual and robot-state evidence in fixed-size latent memory slots and updates them online as the robot acts. Rather than indexing memory by raw time or manually provided task labels, TRACE organizes memory by how the robot reached the current state. It uses *path signatures* [1], a compact representation of the order and size of state changes along an executed trajectory, to summarize the robot's trajectory so far. This allows early evidence to persist through long shared task segments. When the robot later reaches an ambiguous decision point, the policy conditions on TRACE's memory to recover the missing context and select the correct branch. TRACE attaches through a lightweight adapter, allowing it to condition regression-style action-chunking or diffusion policies without changing the policy backbone, action head, or imitation loss.

Concretely, our technical contributions are:

- **Delayed-evidence memory:** We formulate a plug-and-play memory module for visuomotor policies that preserves early task evidence needed for later ambiguous decisions.
- **Path-signature memory routing:** We use path signatures to index fixed-size latent memory by the executed trajectory, enabling order-sensitive historical context without storing long observation windows.
- **Policy integration and evaluation:** We attach TRACE to regression-style action-chunking and diffusion policies through lightweight adapters, and evaluate it on real-world manipulation tasks with diagnostics showing that gains come from preserved historical evidence rather than cues visible only near the decision point.

2 Related Work

Visuomotor policies and memory: Modern imitation policy backbones typically map recent observations to action chunks or action distributions. Regression-style action-chunking policies predict short visuomotor horizons with temporal aggregation [2], while diffusion policies iteratively denoise action sequences [3]. Generalist robot policies scale this paradigm with language conditioning and large cross-embodiment datasets [4, 5, 6, 7, 8, 9, 10]. Despite their architectural differences, these policies remain limited by their conditioning state. Current frames, short observation windows, and language prompts may not preserve the early evidence in the causal history that determines a later behavioral branch. Recurrent policies, transformer windows, and explicit memory systems address partial observability by carrying information forward, including layer-local external memory [11], retrieval-prompt memories for frozen policies [12], declarative scene and episodic memories [13], and keyframe memories for hierarchical imitation [14]. These methods improve temporal context, but often require changing the policy backbone, retrieving from demonstrations, or specializing memory for a planner. TRACE instead maintains a fixed-budget online latent memory within the action policy and changes only the adapter interface. **Path signatures and latent state representations:** Path signatures provide finite-dimensional, order-sensitive summaries of continuous trajectories. They have been used as sequence features, in signature-based control, and as recurrent gating mechanisms such as SigLSTM and SigGRU [1, 15, 16, 17]. A related line of work learns compact latent belief states or world models for prediction, planning, and control under partial observability. Recent systems move beyond pixel prediction toward action-conditioned latent prediction, including

V-JEPA 2 [18], FLARE [19], DreamZero [20], flow-equivariant world models [21], and NextLat, which encourages transformer latents to predict their own next state [22]. TRACE also represents control-relevant state in latent space, but uses path and delta signatures for a different purpose. They are not the policy backbone, a recurrent hidden state, or a learned future-prediction objective. Instead, they serve as trajectory-conditioned routing keys that write visual and state evidence into a fixed-budget external memory, linking memory access to the executed action-conditioned trajectory while leaving the action head and imitation loss unchanged.

3 Problem Setup and Background

Throughout the paper, *history* at time t means the causal execution history that is available online before selecting action a_t. It refers to the executed observations, states, and actions so far.

Delayed-evidence imitation: We study imitation tasks in which a later action depends on evidence observed only earlier in the episode. Each task contains an early cue, a shared execution segment, and a later ambiguous branch point. The cue is visible in the causal history, but the robot later reaches observations that are visually similar across histories while requiring different actions. Here a *branch* is one of the possible task continuations selected by that later action, and the *branch point* is the time when the policy must choose among those continuations.

We consider demonstrations $\mathcal{D} = \{\tau_i\}_{i=1}^{N}$, where $\tau_i = \{(o_t^i, s_t^i, a_t^i)\}_{t=1}^{T_i}$ contains a multi-view observation o_t, robot state s_t, and demonstrated action a_t. Let $x_t = (o_t, s_t)$ and let $\mathcal{H}_t = (x_1, a_1, \ldots, x_{t-1}, a_{t-1}, x_t)$ denote the causal execution history available before choosing the next action. In delayed-evidence tasks, $\mathcal{H}_t$ may contain task-relevant evidence that is no longer present in the current observation window.

For any short observation window of length w, delayed-evidence tasks may contain histories $\mathcal{H}_t$ and $\mathcal{H}'_t$ such that

$$x_{t-w+1:t} \approx x'_{t-w+1:t} \quad \text{but} \quad a_t^\star(\mathcal{H}_t) \neq a_t^\star(\mathcal{H}'_t), \tag{1}$$

where $a_t^\star(\mathcal{H}_t)$ is the demonstrated action required after history $\mathcal{H}_t$. Thus, neither the current frame nor a short window is a sufficient state for imitation once the decisive cue has fallen outside the window. TRACE uses only causal information available during execution, with no cue labels, task labels, future frames, or test-time demonstration retrieval.

Conditional policy interface: Let $z_t = x_{t-w+1:t}$ be the observation window used by a visuomotor policy. We write a conditional imitation backbone as $\hat{y}_t = f_\psi(z_t, c_t)$, where $\hat{y}_t$ may be a single action, a regressed action chunk, an action-token sequence, or a diffusion denoising target. TRACE leaves the backbone, action head, and imitation loss unchanged. It only supplies c_t, a compact summary of the causal history.

Path signatures as streaming trajectory keys: TRACE needs a causal key that captures how the robot reached the current state without storing the full history. Given the robot-state path $S_t : [0,1] \to \mathbb{R}^{d_s}$ up to time t, its depth-p path signature is

$$\mathrm{Sig}_{\le p}(S_t) = \left(1, \int dS, \int_{u_1<u_2} dS_{u_1} \otimes dS_{u_2}, \ldots, \int_{u_1<\cdots<u_p} dS_{u_1} \otimes \cdots \otimes dS_{u_p}\right). \tag{2}$$

Signatures summarize both net changes and ordered interactions among state-coordinate changes, making them sensitive to how a trajectory unfolds rather than only which states it visits [15]. For continuous paths, they are invariant to monotone time reparameterization; in our sampled setting, the piecewise-linear signature provides a descriptor that is less tied to raw execution speed than time-step indexing. Signatures also admit streaming updates, so the trajectory key can be maintained online as new states arrive. In TRACE, signatures are deterministic trajectory keys, not task labels, visual memories, recurrent hidden states, or policy outputs. Visual features store the task evidence, while signatures helping determine where that evidence is written and read. We denote the streamed trajectory signature by $\xi_t = \mathrm{Sig}_{\le p}(S_t)$ and its signature-space increment by $\delta_t = \xi_t - \xi_{t-1}$, with

$\delta_1 = \mathbf{0}$. Appendix A.5 gives signature dimension details, and Appendix A.4 gives the state representation.

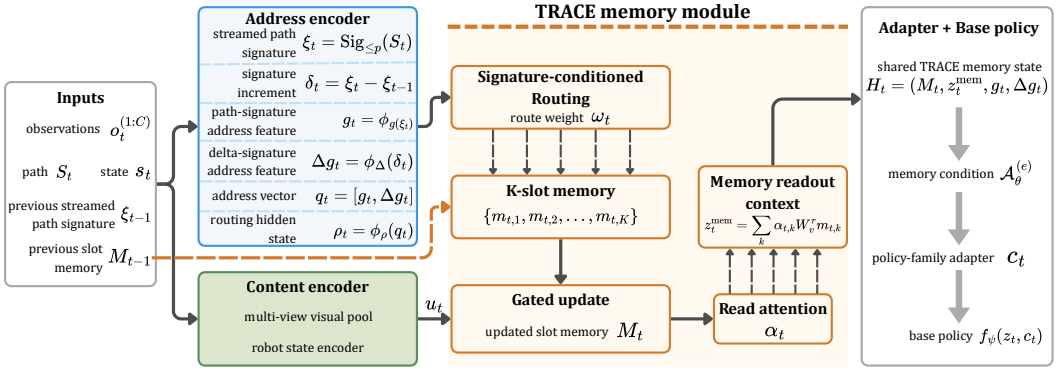

Figure 2: **TRACE signal flow.** TRACE encodes current visual-state evidence as memory content, uses streamed path-signature features as trajectory-derived addresses, updates fixed-size latent memory slots, and returns a compact memory condition to the base visuomotor policy.

4 Methodology

Figure 2 summarizes the TRACE signal flow. TRACE augments an existing visuomotor imitation policy with a fixed-size causal memory. During execution, TRACE writes task-relevant visual and robot-state evidence into latent memory slots. When the current observation later becomes ambiguous, the policy reads this memory as additional context. The base policy keeps its original backbone, action head, action parameterization, and imitation loss; TRACE only changes the context supplied to the policy.

4.1 Causal Memory Interface and Trajectory Keys

Let $z_t = x_{t-w+1:t}$ be the recent observation window used by the base policy, where $x_t = (o_t, s_t)$ contains multi-view observations and robot state. We write the downstream policy as $\hat{y}_t = f_\psi(z_t, c_t)$, where $\hat{y}_t$ is the policy's native prediction target, such as a single action, a regressed action chunk, an action-token sequence, or a diffusion denoising target. TRACE supplies c_t, a compact memory condition carrying causal information that may no longer be visible in z_t.

The central idea is to separate *memory content* from *memory address*. Current visual and robot-state features provide the content to store. The executed robot-state trajectory provides address features for writing and later querying that content. This lets TRACE preserve early evidence through long shared task segments without relying on raw time as the memory index. TRACE constructs the policy condition from a shared memory state $H_t = (M_t, z_t^{\mathrm{mem}}, g_t, \Delta g_t)$ via $c_t = \mathcal{A}_\theta^{(e)}(H_t)$. Here $M_t \in \mathbb{R}^{K \times d}$ is a K-slot latent memory, z_t^{mem} is the vector read from memory, and $g_t, \Delta g_t$ are trajectory-derived address features. The adapter $\mathcal{A}_\theta^{(e)}$ is the only policy-specific component.

To form the address features, let $S_t : [0,1] \to \mathbb{R}^{d_s}$ be the piecewise-linear path of normalized robot states up to time t. TRACE maintains a streamed path signature $\xi_t = \mathrm{Sig}_{\le p}(S_t)$ and its one-step signature increment $\delta_t = \xi_t - \xi_{t-1}$, with $\delta_1 = \mathbf{0}$. The signature ξ_t summarizes the executed trajectory so far, while δ_t reflects recent motion in signature space. TRACE embeds them as $g_t = \phi_g(\xi_t)$ and $\Delta g_t = \phi_\Delta(\delta_t)$. The routing feature is built from the path-signature embedding, the delta-signature embedding when enabled, and an optional first-frame anchor. The path and delta signatures are the trajectory-derived address signals. The optional anchor helps keep addresses stable and is computed from information available at the start of the episode. None of these inputs is a visual label, task label, or hidden branch label.

4.2 Signature-Indexed Slot Memory

TRACE uses the trajectory features above to organize a fixed-size latent memory. At each memory update, it encodes the current visual-state input into a content feature $e_t = \phi_x(x_t)$, where $x_t =$

(o_t, s_t). In implementation, this content is the pooled multi-view visual feature and proprioceptive embedding for the current time step. The feature e_t contains the visual and robot-state evidence currently available to the robot. The address feature q_t is built from g_t and Δg_t, with an optional first-frame anchor when that routing option is enabled. Thus, an early cue can be stored when visible and later exposed through memory when the current image becomes ambiguous. The signature address follows the executed trajectory rather than the clock, while the slot contents carry the visual evidence that distinguishes different histories.

TRACE maintains K memory slots $M_t = \{m_{t,1}, \ldots, m_{t,K}\}$, with $m_{t,k} \in \mathbb{R}^d$. The slots are reset to zero at the start of the history scan or online episode. When slot identities are enabled, a fixed sinusoidal identity η_k is added only inside the addressing path, not stored as recurrent content; otherwise $\lambda_\eta = 0$. To choose a slot address, TRACE maps the address feature to a routing hidden state $\rho_t = \phi_\rho(q_t)$ and compares it with keys computed from the previous slot contents:

$$\bar{m}_{t-1,k} = m_{t-1,k} + \lambda_\eta \eta_k, \qquad \ell_{t,k} = \frac{(W_q \rho_t)^\top W_k \bar{m}_{t-1,k}}{\tau \sqrt{d_r}}, \qquad \omega_{t,k} = \frac{\exp(\ell_{t,k})}{\sum_{j=1}^{K} \exp(\ell_{t,j})}. \tag{3}$$

The routing weight $\omega_{t,k}$ controls how strongly slot k receives the current evidence. The route is computed from signature-derived address features, not from visual labels, task labels, or branch identifiers. TRACE then forms a content proposal from the visual-state evidence, conditioned by the address state only to make the proposal slot-specific:

$$u_t = \phi_w(e_t, q_t), \qquad \tilde{m}_{t,k} = \tanh(\phi_m(\bar{m}_{t-1,k}, u_t, \rho_t)), \qquad \beta_{t,k} = \omega_{t,k}\, \sigma(\phi_\beta(\bar{m}_{t-1,k}, u_t, \rho_t)). \tag{4}$$

Each slot is updated by interpolation, $m_{t,k} = (1 - \beta_{t,k}) m_{t-1,k} + \beta_{t,k} \tilde{m}_{t,k}$. Slots with small routing weights retain their previous contents, while selected slots absorb the new evidence. To condition the policy, TRACE reads from the updated memory using the current visual-state content and the same address state. It forms a read query $u_t^r = \phi_r(e_t, \rho_t)$ and computes

$$\bar{m}_{t,k} = m_{t,k} + \lambda_\eta \eta_k, \qquad a_{t,k} = \frac{(u_t^r)^\top W_k^r \bar{m}_{t,k}}{\sqrt{d}}, \tag{5}$$

$$\alpha_{t,k} = \frac{\exp(a_{t,k})}{\sum_{j=1}^{K} \exp(a_{t,j})}, \qquad z_t^{\mathrm{mem}} = \sum_{k=1}^{K} \alpha_{t,k} W_v^r m_{t,k}. \tag{6}$$

The updated slots M_t, readout z_t^{mem}, and trajectory features $(g_t, \Delta g_t)$ form the shared memory state consumed by the policy adapter.

This differs from simply concatenating signature features to the policy input. Signature-derived features control memory access: they influence where evidence is written, which slots are preserved, and which contents are read back at the branch point. The result is an online, fixed-size memory that stores compact evidence features rather than raw frames, uses no task labels or future observations, and organizes access by the ordered robot-state trajectory.

4.3 Policy Adapter

The previous subsection defines a shared memory state $H_t = (M_t, z_t^{\mathrm{mem}}, g_t, \Delta g_t)$. This state is policy-independent, but different policy families accept extra information through different interfaces. TRACE therefore uses a lightweight adapter as the only policy-specific component:

$$c_t^{(e)} = \mathcal{A}_\theta^{(e)}(H_t) \in \mathcal{C}_e, \tag{7}$$

where $\mathcal{C}_e$ is the memory-input space of policy family e. The adapter only changes how TRACE's memory is passed to the policy. It does not change the memory update, action head, decoder, imitation loss, or use demonstration retrieval at test time.

This keeps TRACE modular. If a policy accepts an additional vector, the adapter compresses the memory readout and trajectory features into that vector. If a policy uses attention over input tokens,

Table 1: Main comparison on real-world delayed-evidence tasks. Values are mean stage progress (%) ± SE.

Method	Tool↑	Book↑	Laundry↑	Cable↑	Medicine↑	Avg. progress↑
Diffusion Policy	18.00 ± 1.41	12.33 ± 0.44	22.00 ± 0.32	34.00 ± 0.48	38.67 ± 1.01	25.00 ± 0.38
ACT	31.00 ± 5.51	13.67 ± 0.49	11.50 ± 2.01	44.00 ± 7.79	27.33 ± 0.05	25.50 ± 1.95
$\pi_{0.5}$	45.50 ± 0.81	51.00 ± 6.99	47.50 ± 1.45	49.00 ± 0.57	59.33 ± 1.45	50.47 ± 1.47
GR00T N1.6	39.00 ± 2.31	12.67 ± 2.14	24.00 ± 2.72	38.00 ± 12.11	39.33 ± 3.29	30.60 ± 2.64
SmolVLA	25.00 ± 1.51	20.00 ± 0.82	12.00 ± 2.16	50.00 ± 1.44	34.67 ± 1.01	28.33 ± 0.65
X-VLA	5.50 ± 0.94	47.00 ± 6.19	41.00 ± 0.36	36.00 ± 21.44	40.00 ± 2.92	33.90 ± 4.51
Ours (Regression)	54.50 ± 17.96	83.00 ± 0.86	81.00 ± 2.84	51.00 ± 1.11	76.67 ± 1.05	69.23 ± 3.65
Ours (Diffusion)	58.50 ± 10.17	68.33 ± 2.86	70.50 ± 0.76	51.00 ± 4.63	49.33 ± 0.17	59.53 ± 2.31

the adapter converts the memory slots, readout, and trajectory features into extra memory tokens. In our regression runs, which use the action-chunking base policy, TRACE passes these memory tokens through the policy's attention interface. In our diffusion runs, the same memory state is converted into an additional conditioning map for the diffusion model. In both cases, the base policy keeps its original action representation and supervised imitation objective. Appendix A.4 gives the shared state and signature settings, Appendix A.8 gives the adapter details, and Appendix A.9 gives the fixed-budget training scan, online update, auxiliary losses, and pseudocode.

5 Experiments

We organize the experiments around four questions. First, whether memory helps delayed-evidence manipulation when the cue is no longer visible at the branch point. Second, whether the gain transfers across policy families. Third, whether TRACE improves over generic history memory. Fourth, whether diagnostics show that TRACE uses earlier evidence rather than only the observation at the decision point. We empirically validate our results on real-world delayed evidence manipulation tasks.

Experiment setup. Figure 3 summarizes part of the delayed-evidence task suite. We evaluate on 5 tasks: *Tool*, where initial object identity determines a later tool sequence; *Book*, where origin determines route and placement; *Laundry*, where origin side determines brush-and-basket versus fold-and-store; *Cable*, where origin side determines the matched device; *Medicine*, where tray origin determines box selection and return. Details of the setup can be found in Appendix A.3.

Figure 3: Overview of the selected delayed-evidence manipulation tasks.

Baselines. We attach TRACE to regression-style action-chunking visuomotor policies and diffusion-based policies. We compare with the corresponding base policies and deployable robot policy families used for imitation or language-conditioned control, including SmolVLA [8], $\pi_{0.5}$ [7], X-VLA [9], and GR00T N1.6 [10]. These VLA baselines are not frozen zero-shot runs, with each is adapted on the same single-task demonstrations and train/eval split. Appendix A.1 gives the model setups. For reproducibility and capacity fairness, Appendix A.2 gives the training hyperparameters and parameter count summary for all baselines and TRACE modules. Following the practice of partial-progress reporting in long-horizon manipulation benchmarks [23, 24], each rollout is scored by stage-level progress over ordered subtasks, including manipulation and memory-dependent decisions, with task-balanced aggregate averages, rollout standard errors, bootstrap uncertainty, and full scoring details in Appendix A.1.

Question 1. Does memory help delayed-evidence manipulation? Yes. Table 1 compares policies that rely on the current observation, fine-tuned VLA-style baselines, and the same policies augmented with TRACE. The rollout rows separate current-frame policies from policies with causal history memory, and the task breakdown shows where start identity determines the later action. Deployment adds only a single streaming signature update, slot update, readout, and adapter call before the unchanged base policy forward pass, with measured overheads reported in Appendix A.8.

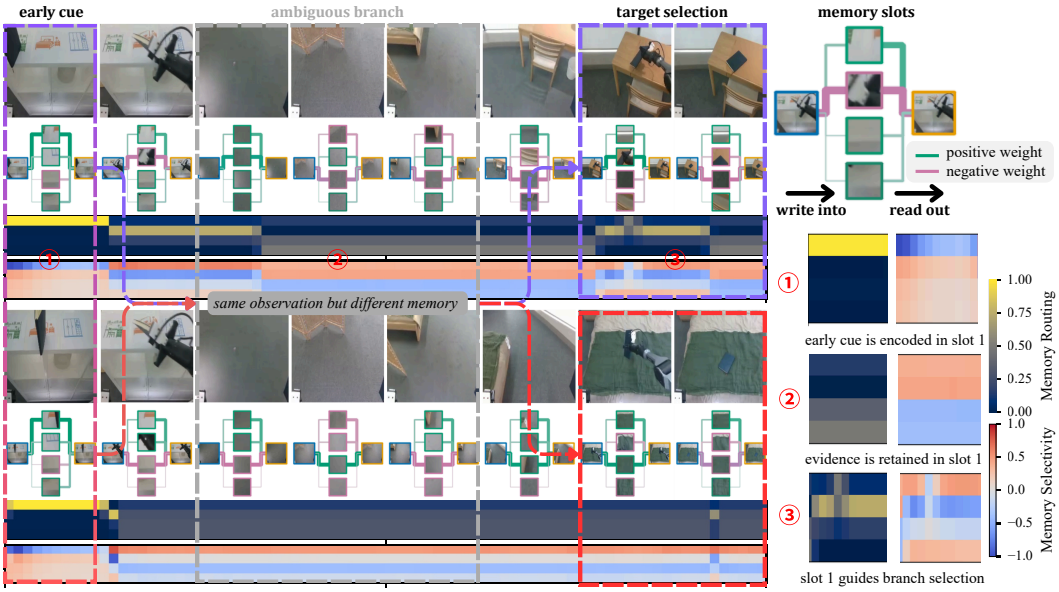

Figure 4: Rollout results for *Book*. The timeline contains early cue, visually ambiguous transit, and target selection, while the overlaid slot graph and right panels show where evidence is written, retained, and read. Positive and negative denote signed memory weights: positive weights add support for the selected slot, whereas negative weights carry opposite-sign evidence that suppresses these slots.

Takeaway. Table 1 shows that TRACE improves the delayed-evidence tasks over policies that do not maintain causal memory, especially when a later decision depends on an origin or object cue that has left view. The failure pattern is not asserted from the aggregate alone. Appendix A.10 ties the common branch errors, their rollout evidence, and the corresponding TRACE behavior to the task cases, while Appendix A.11 separates Tool variance from the main branch-memory comparison.

Question 2. Is the gain confined to a specific policy family? No. Table 1 shows that TRACE improves both the Regression and Diffusion variants while using the same updater, the same signature routing, and the same history scan used in training. Only the policy-facing adapter changes: Regression uses the adapter for the action-chunking regressor, while Diffusion uses the adapter for the diffusion policy. Regression is stronger on average in these measured rollouts. Its memory condition enters in a single forward pass, while the diffusion sampler carries that condition through repeated refinement. The important point is that both policy families benefit from the same TRACE memory state. This supports the modular design claim that TRACE supplies missing history information while leaving the downstream imitation objective and action head intact.

Figure 4 visualizes the measured memory signals. Signature-indexed writes do not collapse to one slot. Routing changes over episode progress, while the readout remains selective for slots carrying history evidence near the branch point. In the Book rollout, the overhead view becomes visually similar after pickup and transit, but the memory graph preserves the origin-dependent trajectory through signed signature scores and high-weight slot edges. Appendix A.10 gives additional task-level case studies that connect these diagnostics to concrete rollout failures and recoveries.

Question 3. Is TRACE better than generic history memory? Yes, when the delayed cue must be preserved through the executed causal history. Table 2 compares TRACE with recurrent, history-token, external-memory, and retrieval alternatives under the same regression base policy, normalization, and evaluation blocks. These controls ask whether delayed-evidence manipulation only needs more history, or whether the history must be organized around the executed causal history.

Takeaway. Generic memory helps relative to no memory, but Table 2 shows that context length or storage capacity alone does not match TRACE. The matched controls can retain history, but they do not explicitly bind the delayed cue to the executed trajectory address that will be read at the branch point. Appendix A.10 connects this distinction to task-level branch errors and TRACE recoveries.

Table 2: Matched memory-module comparison with the regression base policy. The recurrent, transformer-context, LRU, and retrieval-prompt controls are inspired by GRU gating [25], self-attention context [26], least-recently-used external memory access [27], and memory-augmented prompting [12]. Values are mean stage progress percentages ± SE.

Memory module	Online	Fixed	Sig.	Tool	Book	Laundry	Cable	Medicine	Avg.
No memory	–	–	–	31.00 ± 5.51	13.67 ± 0.49	11.50 ± 2.01	44.00 ± 7.79	27.33 ± 0.05	25.50 ± 1.95
GRU recurrent memory	✓	✓	–	40.50 ± 9.12	49.00 ± 3.04	44.00 ± 2.21	47.00 ± 4.98	48.67 ± 1.24	45.83 ± 1.91
Transformer history context	–	–	–	40.50 ± 7.83	61.67 ± 2.54	55.00 ± 1.96	46.00 ± 3.87	57.33 ± 1.41	52.10 ± 1.68
LRU external memory	✓	✓	–	46.50 ± 7.64	57.67 ± 2.48	54.50 ± 2.03	51.00 ± 3.52	60.00 ± 1.22	53.93 ± 1.61
Retrieval-prompt memory	–	–	–	48.00 ± 7.66	62.67 ± 2.18	59.50 ± 1.77	50.00 ± 3.62	61.33 ± 1.12	56.30 ± 1.46
TRACE signature-routed slots	✓	✓	✓	54.50 ± 17.96	83.00 ± 0.86	81.00 ± 2.84	51.00 ± 1.11	76.67 ± 1.05	69.23 ± 3.65

Table 3: Ablations and history-transform diagnostics. Route similarity measures agreement between the transformed online rollout history and the original Full TRACE rollout's slot routing sequence. Branch consistency measures agreement between the branch point memory readout and the branch action. Both diagnostics are normalized to Full TRACE only for cross-task comparison.

Component ablations		History-transform diagnostics			
Variant	Avg.↑	History transform	Tested property	Route similarity↑	Branch consistency↑
Current observation only	25.50 ± 1.95	Time resampling	time-change inv.	92.40 ± 1.10	96.00 ± 0.80
Signature-only	45.50 ± 1.82	Speed jitter	time-change inv.	89.80 ± 1.40	94.70 ± 1.00
Unrouted slot memory	52.17 ± 1.63	State offset	offset inv.	91.20 ± 1.20	95.10 ± 0.90
No-delta routing	61.43 ± 2.21	Sparse sampling	sampling robust.	86.50 ± 1.60	92.30 ± 1.20
Mean readout	62.80 ± 2.44	Order reversal	negative control	37.80 ± 2.50	41.60 ± 2.20
No auxiliary losses	66.10 ± 2.84	Order-preserving controls	summary	89.98 ± 0.68	94.53 ± 0.49
Full TRACE	69.23 ± 3.65	**Full TRACE**	reference	100.00 ± 0.00	100.00 ± 0.00

Question 4. Do diagnostics show that TRACE uses earlier evidence rather than only the observation at the decision point? Yes. Table 3 reports ablations and history-transform diagnostics that test whether TRACE uses the executed history. Route similarity measures whether a transformed online rollout history writes through the same slot route as the original TRACE rollout before the branch point. Branch consistency measures whether the branch-point readout still carries the same delayed branch choice. Time resampling, speed jitter, state offset, and sparse sampling should preserve the relevant history evidence, while order reversal deliberately destroys the causal order of the history. We normalize each raw diagnostic by the corresponding Full TRACE value on the same task and report Full TRACE as 100 only for cross-task comparability. Appendix A.6 gives the full formulas and averaging.

Takeaway. Table 3 supports the intended behavior. The order-preserving transforms keep route similarity and branch consistency close to the Full TRACE reference, while order reversal sharply lowers both diagnostics. This contrast ties the memory readout to ordered historical evidence rather than to the unchanged observation at the decision point.

6 Limitations

The main limitation is the dimensionality of the signature. For standard truncated signatures, the raw feature dimension grows rapidly with the robot-state dimension and signature depth before learned compression. This can increase normalization, projection, memory, and computation costs, and may make TRACE less efficient when higher-dimensional states or deeper signatures are required.

7 Conclusion

TRACE gives visuomotor robot policies a compact causal memory state for the executed history. By routing visual-state writes with path and delta signatures, it stores early task evidence in a fixed-size TRACE memory state and presents that memory through a lightweight adapter. This yields a modular design where memory and action generation remain separate from the base policy objective. Across five real-world delayed-evidence tasks, the same memory module improves both regression and diffusion policy families, and diagnostics show that the gains come from remembering the history rather than relying only on cues visible near the decision point.

References

[1] I. Chevyrev and A. Kormilitzin. A primer on the signature method in machine learning. In *Signature Methods in Finance: An Introduction with Computational Applications*, pages 3–64. Springer, 2025.

[2] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[3] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

[4] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[5] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023.

[6] A. O'Neill, A. Rehman, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, A. Jain, et al. Open X-embodiment: Robotic learning datasets and RT-X models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903. IEEE, 2024.

[7] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[8] M. Shukor, D. Aubakirova, F. Capuano, P. Kooijmans, S. Palma, A. Zouitine, M. Aractingi, C. Pascal, M. Russi, A. Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

[9] J. Zheng, J. Li, Z. Wang, D. Liu, X. Kang, Y. Feng, Y. Zheng, J. Zou, Y. Chen, J. Zeng, et al. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. *arXiv preprint arXiv:2510.10274*, 2025.

[10] J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. Fan, Y. Fang, D. Fox, F. Hu, S. Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

[11] E. Cherepanov, A. K. Kovalev, and A. I. Panov. ELMUR: External layer memory with update/rewrite for long-horizon RL problems. *arXiv preprint arXiv:2510.07151*, 2025.

[12] R. Li, W. Guo, Z. Wu, C. Wang, H. Deng, Z. Weng, Y.-P. Tan, and Z. Wang. MAP-VLA: Memory-augmented prompting for vision-language-action model in robotic manipulation. *arXiv preprint arXiv:2511.09516*, 2025.

[13] M. Lin, X. Liang, B. Lin, L. Jingzhi, Z. Jiao, K. Li, Y. Ma, Y. Liu, S. Zhao, Y. Zhuang, et al. EchoVLA: Robotic vision-language-action model with synergistic declarative memory for mobile manipulation. *arXiv preprint arXiv:2511.18112*, 2025.

[14] T. Buamanee, M. Kobayashi, and Y. Uranishi. Bi-HIL: Bilateral control-based multimodal hierarchical imitation learning via subtask-level progress rate and keyframe memory for long-horizon contact-rich robotic manipulation. *arXiv preprint arXiv:2603.13315*, 2026.

[15] P. Kidger and T. Lyons. Signatory: differentiable computations of the signature and logsignature transforms, on both CPU and GPU. In *International Conference on Learning Representations*, 2021. https://github.com/patrick-kidger/signatory.

[16] M. Rauscher, A. Scagliotti, and F. Pagginelli Patricio. Shortest-path recovery from signature with an optimal control approach. *Mathematics of Control, Signals, and Systems*, 37(2):305–337, 2025.

[17] R. Genet and H. Inzirillo. SigGate: Enhancing recurrent neural networks with signature-based gating mechanisms. *arXiv preprint arXiv:2502.09318*, 2025.

[18] M. Assran, A. Bardes, D. Fan, Q. Garrido, R. Howes, M. Muckley, A. Rizvi, C. Roberts, K. Sinha, A. Zholus, et al. V-JEPA 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

[19] R. Zheng, J. Wang, S. Reed, J. Bjorck, Y. Fang, F. Hu, J. Jang, K. Kundalia, Z. Lin, L. Magne, et al. FLARE: Robot learning with implicit world modeling. *arXiv preprint arXiv:2505.15659*, 2025.

[20] S. Ye, Y. Ge, K. Zheng, S. Gao, S. Yu, G. Kurian, S. Indupuru, Y. L. Tan, C. Zhu, J. Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

[21] H. J. Lillemark, B. Huang, F. Zhan, Y. Du, and T. A. Keller. Flow equivariant world models: Memory for partially observed dynamic environments. *arXiv preprint arXiv:2601.01075*, 2026.

[22] J. Teoh, M. Tomar, K. Ahn, E. S. Hu, P. Sharma, R. Islam, A. Lamb, and J. Langford. Next-latent prediction transformers learn compact world models. *arXiv preprint arXiv:2511.05963*, 2025.

[23] M. Heo, Y. Lee, D. Lee, and J. J. Lim. Furniturebench: Reproducible real-world benchmark for long-horizon complex manipulation. In *Robotics: Science and Systems*, 2023.

[24] O. Mees, L. Hermann, E. Rosete-Beas, and W. Burgard. CALVIN: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics Autom. Lett.*, 7(3):7327–7334, 2022. doi:10.1109/LRA.2022.3180108.

[25] K. Cho, B. Van Merriënboer, Ç. Gulçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1724–1734, 2014.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[27] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR, 2016.

A Technical Appendix

This appendix collects the technical material that supports the main text. The subsections follow the paper narrative. They define the delayed-evidence setup for TRACE, specify the state representation, summarize training hyperparameters, state the signature and memory equations, define diagnostics, and give the adapter, variance, and case study details that are too long for the main paper.

A.1 Problem Setup Details

We consider demonstrations $\mathcal{D} = \{\tau_i\}_{i=1}^{N}$ with

$$\tau_i = \{(o_t^i, s_t^i, a_t^i)\}_{t=1}^{T_i}, \qquad x_t^i = (o_t^i, s_t^i), \qquad \mathcal{H}_t^i = (x_1^i, a_1^i, \ldots, x_{t-1}^i, a_{t-1}^i, x_t^i). \tag{8}$$

Here x_t is the current multi-view observation and robot state, and $\mathcal{H}_t$ is the causal history available before choosing the action at time t. A delayed-evidence task contains an early evidence variable $e_i = \eta(\mathcal{H}_{t_e}^i)$ observed in the history for some $t_e < t_b$. This variable can be the origin shelf, garment side, source tray, object identity, or another cue that determines a later branch. At a branch step $t = t_b$, the branch action a_t is the action whose correct value depends on that earlier evidence.

The central ambiguity is that different early evidence can lead to nearly identical current observations at the branch point. There can be two trajectories with $e_i \neq e_j$ such that

$$x_{t_b}^i \approx x_{t_b}^j, \qquad a_{t_b}^{i,\star} \neq a_{t_b}^{j,\star}, \qquad p(a_{t_b}^\star \mid \mathcal{H}_{t_b}^i) \neq p(a_{t_b}^\star \mid \mathcal{H}_{t_b}^j). \tag{9}$$

A policy of the form $\pi(a_t \mid x_t)$ must assign one action distribution to these ambiguous branch observations, so it is not sufficient for this setting. The needed object is a compact causal history summary $m_t = m(\mathcal{H}_t)$ such that $\pi(a_t \mid x_t, m_t)$ can recover the evidence-dependent branch decision while keeping the interface fixed size.

TRACE implements m_t with an online signature-routed slot memory and the adapter condition derived from it. The early evidence variable e_i is only an unobserved variable used to describe the problem. TRACE does not receive e_i as a supervised label. It also does not receive branch labels, task identifiers, future observations, or future actions. During training and deployment, its memory update at time t reads only the causal history contained in $\mathcal{H}_t$.

Each rollout is scored by decomposing the task into ordered subtasks and then into stages. This metric is inspired by partial-progress reporting in long-horizon manipulation benchmarks, including completed phases or subtasks in FurnitureBench and successful task-chain length in CALVIN-style evaluation [23, 24]. It follows the reporting practice rather than reusing the same formula. For task q, the reported progress is

$$\text{Progress}_q = \frac{100}{N_q T_q S_q} \sum_{i=1}^{N_q} \sum_{k=1}^{T_q} \sum_{j=1}^{S_q} \mathbf{1}\left[\text{stage}_{i,k,j} \text{ succeeds}\right]. \tag{10}$$

Here N_q=25 physical rollouts in the main evaluation, T_q is the number of subtasks, and S_q is the number of stages per subtask. Aggregate averages are task-balanced. Standard errors use rollout-level bootstrap resampling within each task.

Baseline controls. Table 4 gives the baseline protocol referenced in the experiments. The comparison keeps cameras, proprioception, action rate, train split normalization, and held out evaluation lists matched across methods.

Table 4: External VLA baseline protocol. Language conditioned models receive only a generic task prompt.

Baseline	Initialization and trainable parts	Budget	Interface controls
SmolVLA	`lerobot/smolvla_base`, tuned adapter and state mapper, frozen vision	180k updates	3 RGB views, 17-D state, 50-step action chunks
$\pi_{0.5}$	`lerobot/pi05_base` with the released policy adaptation path	180k updates	224px RGB views, 17-D state, 50-step action chunks
X-VLA	`lerobot/xvla-base`, tuned policy transformer and soft prompts	180k updates	Same views and state, 16-step action chunks, no cue prompt
GR00T N1.6	Released N1.6 adaptation with tuned embodiment and action head	180k updates	Same rollout API, action rate, and held out episodes

The baseline protocol table is included to make the comparison auditable rather than to add another performance claim. The important control is that each baseline sees the same camera stream, pro-

prioception layout, action rate, and held-out evaluation list. Language-conditioned methods receive a generic task prompt, so the delayed cue is not leaked through text.

A.2 Model Size and Hyperparameters

Parameter accounting and training hyperparameters are reported to support reproducibility and capacity fairness across policy families. They are not used as standalone performance claims. Table 5 separates total, trainable, and frozen capacity from the local checkpoint and model-cache audit. Table 7 summarizes the training schedules and optimizer settings, while the architectural settings that determine the TRACE module size are listed in Table 10.

Table 5: Model size summary for baselines and TRACE modules from the local checkpoint and model-cache audit.

Model or module	Count scope	Total parameters	Trainable parameters	Frozen parameters or frozen parts	Notes
ACT	Full policy used as the regression base expert	51.62M	51.62M	0	Same observations, state, action rate, and train split as TRACE Regression runs
Diffusion Policy	Full policy used as the diffusion base expert	270.96M	270.96M	0	Same observations, state, action rate, and train split as TRACE Diffusion runs
SmolVLA	Fine tuned LeRobot baseline	450.05M	99.88M	350.17M VLM frozen	Uses the released fine tuning head and matched single task data
$\pi_{0.5}$	Fine tuned LeRobot baseline	3.62B	693.42M	2.92B PaliGemma VLM frozen	Uses the released adaptation path and matched single task data
X-VLA	Fine tuned VLA baseline	879.74M	879.74M	0	Tuned VLM, policy transformer, and soft prompts, with no cue prompt
GR00T N1.6	Fine tuned humanoid policy baseline	2.72B	1.07B	1.66B language/vision frozen	Tuned embodiment interface and action head under the matched rollout API
TRACE updater	Shared signature routed memory updater only	11.91M	11.91M	0	Excludes the base expert and policy specific adapter
Regression adapter	TRACE adapter for the regression base expert	0.79M	0.79M	0	Excludes the shared updater and the base expert
Diffusion adapter	TRACE adapter for the diffusion base expert	16.03M	16.03M	0	Excludes the shared updater and the base diffusion expert

The audit separates model capacity from the memory module. ACT and Diffusion Policy are the base experts for the two TRACE interfaces, while the VLA baselines include their released adaptation paths and frozen components. The TRACE updater is shared across adapters, so the added capacity can be attributed to a fixed memory module plus a small policy-facing translator rather than to a new action backbone.

Table 6 reports the additive parameter budget of TRACE for two policy-facing interfaces. The reference base expert is used only as the audited denominator for the percentage. The row name describes the TRACE interface rather than a fixed combined method. The added count includes the shared updater and the policy-specific adapter. The percentage is computed as $100 \times N_{\text{added}}/N_{\text{base}}$ from the audited counts.

Table 6: Additive TRACE parameter budget by policy-facing interface.

TRACE interface	Reference base expert	Added TRACE parts	Base parameters	Added parameters	Increase	Notes
Regression-policy interface	ACT audit checkpoint	Shared updater plus regression-interface adapter	51.62M	12.69M	24.6%	Interface only
Diffusion-policy interface	Diffusion Policy audit checkpoint	Shared updater plus diffusion-interface adapter	270.96M	27.94M	10.3%	Interface only

The added-parameter table shows that the two interfaces have different capacity costs because they condition different base experts. The regression interface adds 12.69M parameters over the audited ACT checkpoint, and the diffusion interface adds 27.94M over the audited Diffusion Policy check-

point. These counts are used for capacity accounting. They are not used as a substitute for the physical rollout comparisons in Table 1.

Table 7: Training hyperparameters used by the reported protocol. All policies are trained for 180k updates; batch sizes and action windows follow each policy family.

Policy family	Code policy	Updates / batch	Action window	Optimizer and scheduler	Fine-tuning settings
ACT regression	`act`	180k updates; batch 8 or 32	Chunk 100, execute 100	LeRobot ACT optimizer preset; AMP enabled	No TRACE memory; same cameras, state, action normalization, train split, and evaluation API as TRACE Regression
Diffusion Policy	`diffusion`	180k updates; batch 8	2 obs. steps, horizon 104, execute 100, drop last 3 frames	LeRobot Diffusion optimizer preset; AMP enabled	No TRACE memory; same observation and action interface as TRACE Diffusion
SmolVLA	`smolvla`	180k updates; batch 8 or 16	1 obs. step, chunk 50, execute 50, 10 flow steps	AdamW, lr 10^{-4}, weight decay 10^{-10}, grad clip 10; 1000 warmup steps and cosine decay over 30k steps to 2.5×10^{-6}	Frozen vision encoder; train action expert and state projection; generic task prompt only
$\pi_{0.5}$	`pi05`	180k updates; batch 1	1 obs. step, chunk 50, execute 50, 10 inference steps	AdamW, lr 2.5×10^{-5}, weight decay 0.01, grad clip 1; 1000 warmup steps and cosine decay over 30k steps to 2.5×10^{-6}	Frozen vision encoder; train action expert branch; mean/std state and action normalization for the Zeno datasets
X-VLA	`xvla`	180k updates; batch 1	1 obs. step, chunk 16, execute 16, 10 denoising steps	AdamW, lr 10^{-4}, weight decay 0, grad clip 10; 1000 warmup steps and cosine decay over 30k steps to 2.5×10^{-6}	Train policy transformer and soft prompts; no cue prompt
GR00T N1.6	External adaptation	180k updates	Same rollout API and action rate as the robot policies	Released N1.6 adaptation recipe	Tune embodiment interface and action head; language and vision components frozen as in the parameter audit

The training table is intentionally separate from the architecture table below. Table 7 records the optimization budget, action horizon, and fine-tuning choices used in the reported training protocol. Table 10 records the memory architecture values that determine the TRACE module size and routing interface.

A.3 Dataset and Demonstration Details

Table 8 makes the data accounting explicit for the five real robot tasks. The physical rollout count is the main evaluation count used in Equation 10. Tool, Book, and Laundry counts and horizons are audited from the processed LeRobot metadata. Cable and Medicine counts and horizons are audited from the local ROS bag timestamp scan using the same 30 Hz sampling rule as the converter.

Table 8: Dataset scale and demonstration details for the real robot evaluation protocol reported in this paper.

Task	Train demos	Validation demos	Physical eval. rollouts	Episode length or avg. horizon	Subtasks / stages	Held-out factors
Tool	100	0	25	2,910 frames (97.0s)	2/4	Object instances, cue assignments, poses, lighting, distractors
Book	150	0	25	1,392 frames (46.4s)	3/4	Origin cue assignments, poses, routes, lighting, distractors
Laundry	60	0	25	2,220 frames (74.0s)	2/4	Garment instances, origin side assignments, poses, lighting, distractors
Cable	30	0	25	1,770 frames (59.0s)	1/4	Origin side assignments, device poses, lighting, distractors
Medicine	60	0	25	1,759 frames (58.6s)	2/4	Tray origin assignments, box poses, lighting, distractors, bedside props

The dataset table makes the evidence source explicit. Every task uses 25 physical evaluation rollouts, and the reported progress scores in the main table are computed from those rollouts. The train demonstration counts and horizons come from audited metadata or ROS bag timestamp scans. The held-out factors show that the evaluation changes object instances, cue assignments, poses, lighting, and distractors rather than repeating the training episodes.

A.4 State Representation and Normalization

The policy backbones and TRACE use one overhead RGB camera, two wrist RGB cameras, and a 17-D proprioceptive state. The state is ordered as base planar velocity, base yaw velocity, left arm joint positions, and right arm joint positions. Fixed-base tasks keep the same layout and set the base channels to zero before signature computation. The zero mask is applied before train-split standardization, so constant channels contribute no path increments.

At each control step, the masked state is standardized and appended to the streamed history path. The first observed state is used as the basepoint. TRACE omits the scalar signature coordinate. For d_s=17 and p=3, this gives $d_{\mathrm{sig}} = 17 + 17^2 + 17^3 = 5{,}219$. Path signatures and one-step delta signatures are normalized with their own train-split statistics before the learned maps ϕ_g and ϕ_Δ. No cue label, language token, gripper command history, future action, or branch identifier is appended to the signature state.

Table 9: Robot state channels used by TRACE signatures.

Channel block	Dim.	Units	Normalization for signature input	Zeroed channel rule
Base planar velocity (v_x, v_y)	2	m/s	Train mean/std after the zero mask with std floor 10^{-6}	Fixed-base tasks set both channels to 0
Base yaw velocity ω	1	rad/s	Train mean/std after the zero mask with std floor 10^{-6}	Fixed-base tasks set this channel to 0
Left arm joints $q^L_{0,\ldots,6}$	7	rad	Train mean/std, then append to the history path	Never zeroed
Right arm joints $q^R_{0,\ldots,6}$	7	rad	Train mean/std, then append to the history path	Never zeroed
Full state path $\tilde{s}_{1,\ldots,t}$	17	normalized units	Piecewise linear path over masked, standardized states	Constant zero channels add no increments
Path signature ξ_t	5,219	signature units	Train mean/std over history signatures, then $g_t = \phi_g(\xi_t)$	Scalar coordinate omitted
Delta signature δ_t	5,219	signature units	Train mean/std over one-step differences, then $\Delta g_t = \phi_\Delta(\delta_t)$	$\delta_1 = \mathbf{0}$

The state table clarifies what can and cannot enter the signature. The routing key uses only the masked and standardized robot state path, with constant fixed-base channels contributing no path increments. No cue label, branch identifier, future action, or language token is appended. This keeps the memory condition causal and prevents the signature from becoming a hidden branch cue.

Table 10 lists the shared TRACE settings used in the reported runs. These values define the signature interface, memory width, and adapter dimensionality. The downstream policy losses and action heads are unchanged.

Table 10: Core TRACE hyperparameters.

Quantity	Reported setting	Notes
Signature depth p	3	Standard streamed signature
Raw signature dim. d_{sig}	5,219	$17 + 17^2 + 17^3$, scalar term omitted
Signature embedding dims.	512 for g_t, 512 for Δg_t	Shared by both adapters
Slot count K	4 or 6	Chosen by task horizon and branch diversity
Slot width d	512	Matches policy conditioning width
History budget L	24 or 32	Used for training-time causal history reconstruction
History stride r	4 to 8	Covers the recent tail at 30 FPS data rate
Routing hidden size	512	Used for route query and key maps
Adapter hidden size	512	Used by regression memory tokens and diffusion conditioning maps

These hyperparameters keep the shared TRACE updater identical across the two policy families. Depth 3 signatures provide the routed history key, 512-D memory slots match the policy conditioning width, and the fixed history budget controls the supervised training scan. The table also separates these memory settings from the base policy loss and action decoder, which remain unchanged.

A.5 Signature Depth and Invariance

Let S_t be the piecewise-linear interpolation of the causal state history up to time t. TRACE uses the truncated path signature and its first difference

$$\xi_t = \mathrm{Sig}_{\le p}(S_t) \in \mathbb{R}^{d_{\mathrm{sig}}}, \qquad \delta_t = \xi_t - \xi_{t-1}, \qquad g_t = \phi_g(\xi_t), \quad \Delta g_t = \phi_\Delta(\delta_t). \tag{11}$$

The initial delta is $\delta_1 = \mathbf{0}$. The learned maps turn the deterministic signature vectors into routing features.

For an increasing time change α and a constant state offset b, the routing key uses

$$\mathrm{Sig}_{\le p}(S \circ \alpha) = \mathrm{Sig}_{\le p}(S), \qquad \mathrm{Sig}^{S_0+b}_{\le p}(S + b) = \mathrm{Sig}^{S_0}_{\le p}(S). \tag{12}$$

The second identity uses the basepointed signature. The address therefore depends on executed history geometry rather than sampling rate, execution speed, or a constant coordinate shift. The address remains order sensitive, so reversing the causal order changes the key.

For a state path with dimension d_s and standard truncated signatures with the scalar coordinate omitted,

$$d_{\mathrm{sig}}(p) = \sum_{k=1}^{p} d_s^k. \tag{13}$$

The dimension grows exponentially with depth. With the 17-D state path used in our experiments, the jump from $p{=}3$ to $p{=}4$ increases the raw signature from $5{,}219$ to $88{,}740$ coordinates before learned compression.

Table 11: Signature depth scaling for $d_s{=}17$.

Depth p	Standard d_{sig}	Log signature dim.	Used in reported runs	Practical interpretation
1	17	17	No	Captures net displacement only
2	306	153	No	Adds pairwise order terms at low cost
3	5,219	1,785	Yes	Captures route and phase interactions while remaining practical for learned compression
4	88,740	22,593	No	Requires aggressive compression before routing

We use $p{=}3$ as a practical balance. Depth 1 is close to a displacement descriptor and loses much of the ordering that distinguishes delayed branches. Depth 2 is cheaper but may underrepresent multi-stage histories such as cue, transit, and branch setup. Depth 4 substantially increases feature storage and learned-compression cost, making compression the dominant TRACE term.

Log signatures are attractive because they remove algebraic redundancies and reduce the feature dimension, as shown in Table 11. They are also less redundant as inputs to a learned map. The tradeoff lies in engineering and numerical complexity. Streamed online updates, delta features, normalization statistics, and GPU support must match the deployment path. We therefore use standard streamed signatures in all reported experiments. Log signatures are an alternative focused on efficiency and remain outside the scope of these results.

A.6 Diagnostic Metric Definitions

For each held-out online evaluation rollout e from task q and history transform T, we run a causal diagnostic pass along the transformed history and compare it with the identity pass on the same online rollout. Let I denote the identity transform, $t_b(e)$ the first ambiguous branch point, and $\mathcal{P}_e = \{t : t < t_b(e)\}$ the history indices before the branch point. The diagnostic inputs are the slot routing distributions $\omega_t^{T,e} \in \Delta^{K-1}$ for $t \in \mathcal{P}_e$, the branch point readout $z_{t_b(e)}^{T,e} \in \mathbb{R}^d$, and the branch action label $\hat{b}^{T,e}$ induced by the action head. The transform is applied only to the robot state path that produces TRACE signatures. The rollout identity, branch frame, branch label, and visual observations used by the policy come from the online execution and are unchanged, so the comparison isolates the effect of the transformed history route while preserving the online-rollout data source.

Route similarity measures whether the transformed history writes through the same memory slots as the identity online rollout pass before the branch decision. For a task q with held out rollouts $\mathcal{E}_q$, the raw route score is

$$r(e, T) = |\mathcal{P}_e|^{-1} \sum_{t \in \mathcal{P}_e} \frac{\langle \omega_t^{T,e}, \omega_t^{I,e} \rangle}{\|\omega_t^{T,e}\|_2 \, \|\omega_t^{I,e}\|_2}, \tag{14}$$

$$R_q(T) = |\mathcal{E}_q|^{-1} \sum_{e \in \mathcal{E}_q} r(e, T). \tag{15}$$

The routing vectors are nonnegative probability distributions, so the cosine similarity lies in $[0, 1]$. High route similarity means that the transformed history follows the same sequence of soft slot

addresses as the original Full TRACE rollout. The reported value is normalized to the Full TRACE identity online rollout pass on the same task,

$$\mathrm{RouteSim}_q(T) = 100\,\frac{R_q(T)}{R_q(I)}. \tag{16}$$

For the Full TRACE reference, $R_q(I) = 1$ by construction. We keep the denominator explicit because all task level scores are reported relative to this reference before averaging across tasks.

Branch consistency measures whether the branch point evidence read from memory is preserved and whether it supports the same delayed branch action. We compute a continuous readout term and a discrete action agreement term,

$$C_q^{\mathrm{read}}(T) = |\mathcal{E}_q|^{-1} \sum_{e \in \mathcal{E}_q} \frac{1 + \frac{\langle z^{T,e}_{t_b(e)}, z^{I,e}_{t_b(e)} \rangle}{\|z^{T,e}_{t_b(e)}\|_2 \, \|z^{I,e}_{t_b(e)}\|_2}}{2}, \tag{17}$$

$$C_q^{\mathrm{act}}(T) = |\mathcal{E}_q|^{-1} \sum_{e \in \mathcal{E}_q} \mathbf{1}\left[\hat{b}^{T,e} = \hat{b}^{I,e}\right], \tag{18}$$

$$B_q(T) = \frac{1}{2}\left(C_q^{\mathrm{read}}(T) + C_q^{\mathrm{act}}(T)\right), \tag{19}$$

$$\mathrm{BranchCons}_q(T) = 100\,\frac{B_q(T)}{B_q(I)}. \tag{20}$$

The readout cosine is mapped from $[-1, 1]$ to $[0, 1]$, and the action term is the agreement rate with the original Full TRACE branch action. High branch consistency means that the transformed history exposes a similar memory readout at the first ambiguous decision and leads the action head to choose the same branch. The final table values are task balanced averages, $|\mathcal{Q}|^{-1} \sum_q \mathrm{Metric}_q(T)$. Standard errors are computed by bootstrap resampling held out rollouts within each task and recomputing the task balanced average.

Order reversal is used as a negative control because it preserves the held out episode, the branch frame, and many marginal state values, but it destroys the causal order of the history. This is exactly the information that path signatures and the slot routing keys are meant to encode. Order-preserving transforms such as time resampling or speed jitter keep both diagnostics high, while reversal reduces them when TRACE uses ordered history evidence rather than cues visible only near the decision point or the same states without their order. Full TRACE is the identity online rollout reference, so both normalized diagnostics are reported as 100 for I by construction. These quantities are computed only after training. They are not losses, they are not used for model selection, and no gradient is taken through them.

A.7 Long-Horizon Memory Stability

The invariance diagnostics above test whether the branch decision depends on ordered history evidence. We also run a long-history stability pass to check whether the fixed slot memory remains usable as held-out online rollouts provide longer causal histories before the branch readout. This pass is a diagnostic of the reported horizons, not a theoretical guarantee for arbitrary episode length.

The protocol has two forms. In the reported-horizon online-history pass, each held-out physical rollout supplies the causal history. The trained updater is applied along that executed history, and the memory state is recorded at the first ambiguous branch point. In extended-history diagnostics, we continue the same online-history accumulation with measured causal segments from recorded online rollouts and deployed diagnostic passes. The extensions use repeated distractor segments, speed jitter, sparse resampling, and additional shared-route online segments when those segments are available in the recorded diagnostic source. The branch frame, branch label, visual observations, and policy weights are held fixed, so changes in the diagnostic scores reflect changes in the memory route and stored history evidence. These diagnostic rows are not extrapolated rollout success rates.

For long-history stability, we record slot churn, write entropy, slot occupancy, branch readout consistency, and branch decision consistency. For an episode with write distributions $\omega_1, \ldots, \omega_T$, these are

$$\text{Churn} = (T-1)^{-1} \sum_{t=2}^{T} \mathbf{1}\left[\arg\max_j \omega_t^{(j)} \neq \arg\max_j \omega_{t-1}^{(j)}\right], \tag{21}$$

$$\text{WriteEnt} = (T \log K)^{-1} \sum_{t=1}^{T} \left(-\sum_{j=1}^{K} \omega_t^{(j)} \log \omega_t^{(j)} \right), \tag{22}$$

$$\text{Occupancy} = K^{-1} \sum_{j=1}^{K} \mathbf{1}\left[\exists t \leq T,\ j = \arg\max_{j'} \omega_t^{(j')}\right]. \tag{23}$$

Branch readout consistency is the cosine similarity between the branch readout from the extended history and the readout from the matched reported-horizon online-history pass. Branch decision consistency is the corresponding action agreement rate.

Table 12: Long-horizon memory stability diagnostics for fixed slot TRACE memory. Values are task-balanced means over held-out rollouts. Readout and decision consistency compare each extension with the matched reported-horizon branch readout.

Diagnostic pass	History extension	Slot churn↓	Write entropy↓	Slot occupancy↑	Readout cons.↑	Decision cons.↑
Reported-horizon online history	$1.0\times$	0.010	0.356	0.646	1.000	1.000
Extended history	$1.25\times$	0.017	0.374	0.690	0.982	0.968
Extended history	$1.5\times$	0.025	0.397	0.710	0.962	0.936
Extended history	$2.0\times$	0.041	0.431	0.744	0.928	0.904
Repeated distractor extension	$1.5\times$	0.052	0.462	0.758	0.907	0.872
Shared-route online extension	$2.0\times$	0.035	0.412	0.737	0.943	0.916

The stability readout is a measured diagnostic over the evaluated extensions rather than a proof for arbitrary horizons. Across the evaluated extensions, slot churn stays near 0.05 or lower, occupancy remains broad but noncollapsed, and branch decision consistency remains at least 0.872 even under repeated distractor segments. The larger entropy under the distractor extension comes from the added segment's reuse of ambiguous shared-route observations, while the branch readout remains close to the reported-horizon reference. Rising churn or entropy together with falling branch consistency is the overwrite or diffuse-write failure signature. The diagnostic therefore supports the memory claims only over the evaluated horizons and history extensions. It does not show that fixed slots avoid overwriting old information for arbitrarily long rollouts.

A.8 Adapter Architectures

Both TRACE variants use the same signature-indexed memory updater. The main text names the two variants by their base objectives, Regression and Diffusion. In this architecture subsection, the same variants are described as the regression adapter and diffusion adapter because the figure and table refer to the policy-facing modules that present the memory readout to each base policy. For a policy family indexed by e,

$$H_t = (M_t, z_t^{\text{mem}}, g_t, \Delta g_t), \qquad \mathcal{A}_\theta^{(e)} : \mathcal{H}_{\text{TRACE}} \to \mathcal{C}_e, \qquad c_t^{(e)} = \mathcal{A}_\theta^{(e)}(H_t). \tag{24}$$

Here $\mathcal{C}_e$ is the native conditioning space of the downstream policy family. The adapter changes the policy input condition, but it leaves the action parameterization and imitation objective unchanged.

Table 13: Adapter interfaces for the regression and diffusion backbones.

Adapter	TRACE inputs	Adapter computation	Expert conditioning produced	Expert loss
Regression adapter	Slot memory $M_t \in \mathbb{R}^{K\times 512}$, readout z_t^{mem}, signature embeddings $g_t, \Delta g_t$	Map each slot with W_M. Map $[z_t^{\mathrm{mem}}, g_t, \Delta g_t]$ into one summary token. Optionally use the summary for feature modulation	512-D memory tokens concatenated to the regression policy attention memory. The action head regresses the native action chunk	Unchanged chunk L1 loss plus the standard optional KL term
Diffusion adapter	Same $M_t, z_t^{\mathrm{mem}}, g_t, \Delta g_t$ from the shared updater	Pool slots with readout attention weights. Concatenate $\mathrm{pool}(M_t)$, z_t^{mem}, g_t, and Δg_t. Pass the result through a zero-initialized MLP	512-D additive global conditioning vector appended to time-step and action conditioning. Diffusion denoising iterations are unchanged	Unchanged diffusion denoising loss over the action trajectory
Shared updater	Masked state path, pooled multi-view visual feature v_t, proprioceptive embedding p_t, and previous slots M_{t-1}	Compute $g_t, \Delta g_t$. Route a write distribution ω_t. Update gated slot candidates and read slots with the current visual-state query	Policy-agnostic TRACE memory state H_t consumed by either adapter	Auxiliary balance, entropy, and consistency losses only during training

The adapter comparison shows where the two TRACE variants differ. Both consume the same routed memory state, but the regression adapter exposes it as attention memory tokens and the diffusion adapter exposes it as a global conditioning vector. This is why the main experiments can attribute shared gains to the causal memory while still allowing each policy family to keep its own action loss.

Table 14 separates the unchanged base policy forward pass from TRACE's per-step computation. Entries are mean / p95 latency. The regression row is measured over 1,900 post-warmup control steps from the streaming regression policy profile on an NVIDIA GeForce RTX 5090 with CUDA 12.8 and PyTorch 2.9.1. The diffusion row combines the measured shared TRACE path with recorded diffusion and adapter timings from the deployed TRACE-conditioned policy.

Table 14: Per-step deployment latency breakdown in milliseconds.

Policy path	Base forward	Signature	Slot update	Readout	Adapter	TRACE overhead	Total loop	Overhead
Regression adapter	5.90 / 7.21	0.52 / 0.69	0.90 / 1.26	0.58 / 0.74	0.10 / 0.15	2.11 / 2.66	8.32 / 10.02	35.8%
Diffusion adapter[†]	118.00 / 142.00	0.52 / 0.69	0.90 / 1.26	0.58 / 0.74	0.14 / 0.21	2.15 / 2.72	120.46 / 144.87	1.8%

[†] Diffusion timing uses the shared measured TRACE path. Base forward is the 100-step diffusion denoising call.

The latency table shows that TRACE adds a small fixed computation before the unchanged base policy call. For the regression adapter, the overhead is visible because the base policy is already fast. For the diffusion adapter, the same memory path is small compared with the 100-step diffusion denoising call. The table therefore supports the claim that TRACE is an online conditioning module rather than a second policy pass or an offline retrieval procedure.

A.9 Memory Update, Training, and Online Inference

At each step, camera features are pooled as $v_t = C^{-1}\sum_c \rho(f_{\mathrm{vis}}(o_t^{(c)}))$, and the proprioceptive embedding is $p_t = \phi_s(\tilde{s}_t)$. These features form the visual-state content $e_t = [v_t, p_t]$. The address feature q_t is built from the projected path signature and delta signature. When first-frame anchoring is enabled, q_t also includes the causal first-frame anchor. With this address vector, the route query and write distribution are

$$\rho_t = \phi_\rho(q_t), \qquad \bar{m}_{t-1,k} = m_{t-1,k} + \lambda_\eta \eta_k, \tag{25}$$

$$\ell_{t,k} = \frac{(W_q\rho_t)^\top W_k \bar{m}_{t-1,k}}{\tau\sqrt{d_r}}, \qquad \omega_{t,k} = \frac{\exp(\ell_{t,k})}{\sum_{j=1}^K \exp(\ell_{t,j})}. \tag{26}$$

where η_k is a fixed sinusoidal slot identity when enabled; setting $\lambda_\eta = 0$ gives the adapters that do not use slot identity. The content proposal, write strength, and slot update are

$$u_t = \phi_w(e_t, q_t), \tag{27}$$

$$\tilde{m}_{t,k} = \tanh(\phi_m(\bar{m}_{t-1,k}, u_t, \rho_t)), \tag{28}$$

$$\beta_{t,k} = \omega_{t,k}\sigma(\phi_\beta(\bar{m}_{t-1,k}, u_t, \rho_t)), \tag{29}$$

$$m_{t,k} = (1-\beta_{t,k})m_{t-1,k} + \beta_{t,k}\tilde{m}_{t,k}. \tag{30}$$

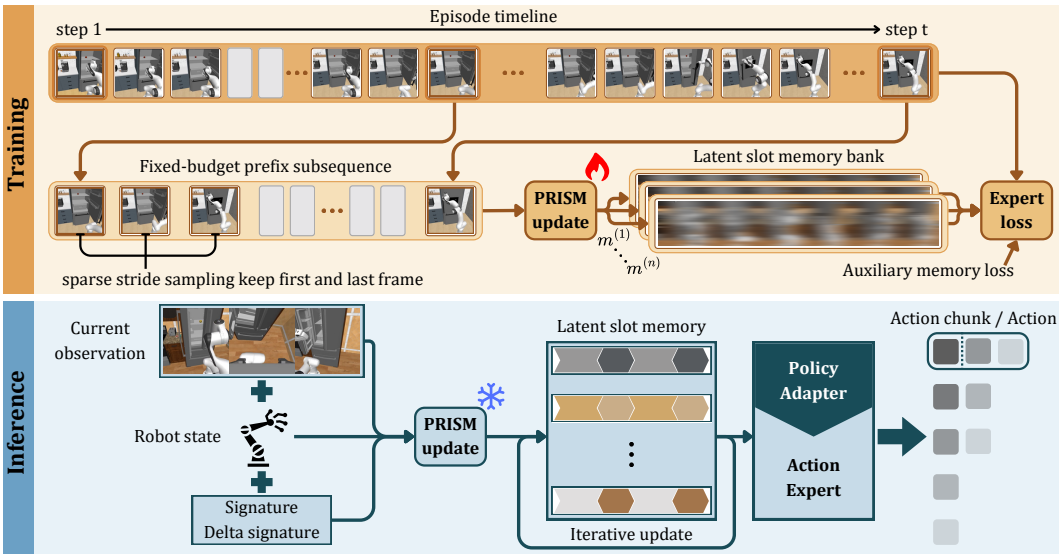

Figure 5: Training and inference consistency. Training scans the masked fixed-budget history available online, and deployment applies the same updater once per executed step.

The memory readout uses a separate attention distribution:

$$u_t^r = \phi_r(e_t, \rho_t), \tag{31}$$

$$\bar{m}_{t,k} = m_{t,k} + \lambda_\eta \eta_k, \tag{32}$$

$$a_{t,k} = \frac{(u_t^r)^\top W_k^r \bar{m}_{t,k}}{\sqrt{d}}, \tag{33}$$

$$\alpha_{t,k} = \frac{\exp(a_{t,k})}{\sum_{j=1}^{K} \exp(a_{t,j})}, \tag{34}$$

$$z_t^{\mathrm{mem}} = \sum_{k=1}^{K} \alpha_{t,k} W_v^r m_{t,k}. \tag{35}$$

During training, each target time t uses a padded fixed-budget causal history index sequence $I_{L,r}(t) = (i_1, \ldots, i_L) \subseteq \{1, \ldots, t\}$ with valid mask b_l. The selected valid entries contain the first available frame, the current frame, and a recent stride-sampled tail. Starting from $M^{(0)} = \mathbf{0}$, the scan is

$$M^{(l)} = \mathcal{U}_\theta(M^{(l-1)}, o_{i_l}, s_{i_l}, q_{i_l}, b_l), \qquad l = 1, \ldots, L. \tag{36}$$

This scan applies the same updater that deployment uses once per online step with the cached memory state. The final scanned memory forms H_t, and the native policy loss for expert family e is

$$\mathcal{L}_{\mathrm{policy}}^{(e)} = \mathbb{E}_{(\tau,t)\sim\mathcal{D}} \left[\ell_e \left(\mathcal{E}_\psi^{(e)}(x_t, \mathcal{A}_\theta^{(e)}(H_t)), y_t^\star \right) \right]. \tag{37}$$

The finite-slot updater has three common failure modes during training. Slot collapse writes most evidence through a few slots. Diffuse writing spreads one observation across many slots, making later addresses weak. Read-write mismatch stores information in a form that the readout cannot recover after subsequent updates. TRACE uses one lightweight stabilizer for each case, then keeps the downstream imitation objective unchanged.

$$\mathcal{L} = \mathcal{L}_{\mathrm{policy}}^{(e)} + \lambda_{\mathrm{bal}} \mathcal{L}_{\mathrm{bal}} + \lambda_{\mathrm{ent}} \mathcal{L}_{\mathrm{ent}} + \lambda_{\mathrm{cons}} \mathcal{L}_{\mathrm{cons}}. \tag{38}$$

For the l-th selected history element of target t, let $\omega_{t,l}$ be the slot-routing distribution, $u_{t,l}$ the write proposal, and $z_{t,l}^{\mathrm{mem}}$ the memory readout. With $N_v = \sum_{t,l} b_{t,l}$ and $\bar{\omega}^{(j)} = N_v^{-1} \sum_{t,l} b_{t,l} \omega_{t,l}^{(j)}$, the

auxiliary terms are

$$\mathcal{L}_{\text{bal}} = K^{-1} \sum_{j=1}^{K} \left(\bar{\omega}^{(j)} - K^{-1} \right)^2, \tag{39}$$

$$\mathcal{L}_{\text{ent}} = (N_v \log K)^{-1} \sum_{t,l} b_{t,l} \left(- \sum_{j=1}^{K} \omega_{t,l}^{(j)} \log \omega_{t,l}^{(j)} \right), \tag{40}$$

$$\mathcal{L}_{\text{cons}} = N_v^{-1} \sum_{t,l} b_{t,l} d^{-1} \left\| \tanh z_{t,l}^{\text{mem}} - \tanh u_{t,l} \right\|_2^2. \tag{41}$$

$\mathcal{L}_{\text{bal}}$ penalizes uneven average routing, so it discourages slot collapse. $\mathcal{L}_{\text{ent}}$ penalizes high-entropy routing at each valid write step, so it discourages diffuse writing. $\mathcal{L}_{\text{cons}}$ aligns the bounded memory readout with the current write proposal, so it reduces read-write mismatch.

These terms are finite-capacity stabilizers rather than theoretical guarantees. They bias training toward balanced, addressable, and readable writes, but they do not prove that every piece of causal information remains recoverable. Since the memory has a fixed number of slots and a bounded slot width, sufficiently long horizons or trajectories with more pieces of information than the memory can represent may still force multiple facts to share capacity. In such regimes, later updates can overwrite or blur earlier content, especially when evidence is repeatedly routed to nearby slots or when visually similar observations require different interpretations.

The auxiliary terms act only on memory. The downstream policy loss remains chunk L1 with the standard optional KL term for Regression, and the native diffusion denoising loss for Diffusion.

Algorithm 1 Causal TRACE memory update and adapter conditioning.

Require: Observation o_t, robot state s_t, previous memory M_{t-1}, previous signature ξ_{t-1}, optional first-frame anchor a^0, state mask Z_q, train-split normalization statistics, policy family $e \in \{\text{regression}, \text{diffusion}\}$.
Ensure: Updated memory M_t, signature ξ_t, adapter conditioning c_t, and policy prediction $\hat{y}_t$.

1: Apply the task mask and state normalization.

$$\bar{s}_t = Z_q(s_t), \qquad \tilde{s}_t = (\bar{s}_t - \mu_s)/(\sigma_s + \epsilon).$$

2: Append $\tilde{s}_t$ to the piecewise linear history path S_t.
3: Compute $\xi_t = \text{Sig}_{\leq p}(S_t)$ and $\delta_t = \xi_t - \xi_{t-1}$.
4: Remove the scalar signature coordinate, standardize ξ_t and δ_t, then map them to $g_t = \phi_g(\xi_t)$ and $\Delta g_t = \phi_\Delta(\delta_t)$.
5: Encode the current observation as $v_t = C^{-1} \sum_c \rho(f_{\text{vis}}(o_t^{(c)}))$ and $p_t = \phi_s(\tilde{s}_t)$.
6: Compute address features $q_t = [g_t, \Delta g_t]$, appending a^0 only when first-frame anchor routing is enabled, and set $\rho_t = \phi_\rho(q_t)$.
7: Route the write with Equation 26.
8: Form gated write candidates from visual-state content and address state, then update slots with Equation 30.
9: Read memory with Equation 35.
10: Construct $H_t = (M_t, z_t^{\text{mem}}, g_t, \Delta g_t)$ and $c_t^{(e)} = \mathcal{A}_\theta^{(e)}(H_t)$.
11: Predict $\hat{y}_t = f_\psi(z_t, c_t^{(e)})$ with the unchanged base policy.

A.10 Additional Case Studies

Table 15 summarizes the concrete delayed-evidence structure behind the five tasks. These case studies connect the diagnostic quantities in the main paper to rollout-level behavior and give the evidence source for the baseline failure patterns discussed in the main text.

Table 15: Additional task-level case studies linking baseline failures, evidence, and TRACE behavior.

Task	History evidence	Ambiguous branch	Baseline failure	Failure evidence	TRACE behavior
Tool	Initial object identity determines the later tool sequence	Object and tool are no longer jointly visible when the sequence must branch	Wrong tool order at the branch, or correct branch followed by contact loss	Table 1 shows high Tool variance, and Table 18 separates delayed-memory errors from contact losses	Stores the object-dependent history and improves branch selection, while contact-rich stages still create variance
Book	Origin shelf determines route and placement	After pickup and transit, the overhead view is similar across origins	Places on a visually plausible target that is inconsistent with the origin route	Figure 4 shows similar views after transit, and Table 2 shows the matched memory gap	Reads the origin-dependent slot near placement and preserves route-specific evidence
Laundry	Origin side determines brush-and-basket versus fold-and-store	Garment pose becomes visually similar after the shared carry segment	Executes the visually dominant routine regardless of origin side	Tables 1 and 2 show the largest gains on this origin-side branch task	Routes the side cue during pickup and reuses it at the later branch
Cable	Origin side determines the matched device	Traversal and local device pose provide evidence still visible near the device choice	Reaches the workspace but can choose the wrong device when local geometry is ambiguous	Tables 1 and 2 show closer scores, consistent with partial evidence from current geometry	Memory helps at the device choice, but local manipulation still contributes many credited stages
Medicine	Tray origin determines box selection and return	Boxes enter a shared bedside area before the return decision	Selects or returns the wrong box after a correct early pickup	Tables 1 and 2 show gains on the tray-origin branch, and Table 3 supports history-sensitive readout	Keeps the tray-origin cue available through the shared staging segment

The case studies explain why the same memory module has different task-level effects. Book, Laundry, and Medicine place much of the later score on remembering an origin cue, so TRACE changes the dominant branch error pattern. Cable still contains useful local geometry near the device choice, which narrows the gap, while Tool combines the delayed branch with contact-heavy execution and therefore needs the separate variance analysis in Appendix A.11.

A.11 Variance and Failure Analysis

The Tool task has the highest standard error in Table 1. Its progress score combines a discrete delayed memory decision with several contact-rich manipulation stages after the decision. A rollout can remember the correct object and tool branch but lose many credited stages due to grasp slip, weak tool contact, or recovery timeout. A wrong delayed branch can also remove several downstream stages at once. This explains why Tool is noisier than Book, Laundry, and Medicine, where the origin cue more directly determines the branch score.

Table 16 reports a simple robustness check for the main averages. The drop-Tool column removes the task with the largest SE and recomputes the task-balanced mean from the remaining four task means. TRACE Regression remains the strongest method at 72.92, and TRACE Diffusion remains above the strongest non-TRACE baseline at 59.79 versus 51.71. Using only the reported task SEs gives drop-Tool 95% intervals of $[71.28, 74.55]$ for TRACE Regression, $[57.10, 62.48]$ for TRACE Diffusion, and $[48.13, 55.29]$ for the strongest non-TRACE baseline. This check uses no Tool rollouts, so the average gain is not carried by the high-variance task. The table is computed directly from Table 1.

Table 16: Leave-one-task-out task-balanced averages for the main comparison. Values are recomputed from the task means in Table 1.

Method	All tasks	Drop Tool	Drop Book	Drop Laundry	Drop Cable	Drop Medicine
Diffusion Policy	25.00	26.75	28.17	25.75	22.75	21.58
ACT	25.50	24.12	28.46	29.00	20.88	25.04
$\pi_{0.5}$	50.47	51.71	50.33	51.21	50.83	48.25
GR00T N1.6	30.60	28.50	35.08	32.25	28.75	28.42
SmolVLA	28.33	29.17	30.42	32.42	22.92	26.75
X-VLA	33.90	41.00	30.62	32.12	33.38	32.38
Ours (Regression)	**69.23**	**72.92**	**65.79**	**66.29**	**73.79**	**67.38**
Ours (Diffusion)	59.53	59.79	57.33	56.79	61.66	62.08

Table 17 separates the noisy Tool score from the aggregate conclusion. Tool has a wide interval for both TRACE variants because a discrete delayed branch is followed by several contact-rich stages. The same table also reports the drop-Tool averages and their gaps against the strongest non-TRACE baseline, using the reported task SEs from Table 1. Regression keeps a 21.21 point drop-Tool advantage over $\pi_{0.5}$, and Diffusion keeps an 8.08 point advantage.

Table 17: Tool-task uncertainty and drop-Tool robustness for TRACE variants. Tool intervals use the reported mean ± 1.96 SE. Drop-Tool intervals are computed from the reported task SEs in Table 1.

Method	Tool mean $\pm$ SE	Tool 95% interval	Drop-Tool avg.	Drop-Tool gap vs. $\pi_{0.5}$
Ours (Regression)	54.50 ± 17.96	$[19.30, 89.70]$	$72.92\ [71.28, 74.55]$	$+21.21\ [17.27, 25.15]$
Ours (Diffusion)	58.50 ± 10.17	$[38.57, 78.43]$	$59.79\ [57.10, 62.48]$	$+8.08\ [3.60, 12.56]$

Table 18 gives the failure categories used to interpret the high Tool variance. The categories separate whether TRACE reaches the delayed branch with the correct object-conditioned decision from later contact-rich losses. This distinction matters because Tool can produce mid-range or low progress for different reasons. Some rollouts preserve the delayed cue but lose downstream stages through contact, while others fail the memory-dependent branch itself.

Table 18: Tool failure taxonomy for interpreting the high-variance rollouts. The categories separate delayed-memory failures from contact and recovery failures after the branch.

Failure class	Operational definition	Score pattern	Interpretation
Correct branch with contact loss	The object-dependent tool order is selected correctly, then progress drops from grasp slip, weak contact, or incomplete tool engagement	Shared setup and branch stages receive credit, but later manipulation stages are missing	Memory succeeds, while execution noise lowers total progress
Wrong branch or tool order	The shared setup reaches the delayed decision, but the selected tool sequence does not match the initial object identity	Early stages receive credit, followed by a sharp loss at the branch and its downstream stages	Direct delayed-evidence error. This is the failure mode TRACE is designed to reduce
Recovery timeout	Manipulation stalls during recovery after a partial success and exceeds the timeout	Progress plateaus after partial later-stage credit	Separates memory retention from long-horizon recovery and contact robustness
Pre-branch setup failure	The rollout fails before the delayed decision can be evaluated	Low progress before the branch point	Does not test delayed memory, but still contributes to rollout-level Tool variance
Other or unclassified	The score trace or video evidence does not cleanly match the categories above	Mixed or inconsistent stage evidence	Reserved for audit before making branch-level rate claims