CAUSALWAM: MECHANISM-FIRST CAUSAL WORLD-ACTION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Interacting with devices requires reasoning about hidden modes, accumulated progress, and delayed effects that determine whether and when actions succeed. Neural and programmatic world models provide useful foundations, but jointly supporting hidden-state inference, compositional planning, and local rule revision remains challenging. We introduce CausalWAM, a mechanism-first world-action model built around executable local causal programs. Each program specifies typed variables, preconditions, transition rules, and editable parameters. The model infers a mixed discrete–continuous belief from interaction history, composes program effects with subsequent preconditions, and grounds selected operations into action chunks. An external library stores reusable programs; local edits are validated against observed transitions and attached to affected device instances. We introduce MechanismBench to measure hidden-state tracking, alternate-action prediction, held-out composition, and adaptation after unannounced mechanism changes. CausalWAM achieves 76.2% success on held-out compositions and 85.4% success by the tenth post-change episode, outperforming supervision-matched, programmatic, and multimodal-agent baselines. Controlled diagnostics examine program execution and selective repair; a Franka robot uses the same interface to operate hidden-mode, accumulator, and delayed-trigger fixtures.

1 INTRODUCTION

Operating a device requires choosing actions that satisfy its internal control logic. A button press may advance a hidden mode, sustained input may accumulate progress, and an earlier action may enable an effect only after a delay. Two interaction histories can therefore produce the same visible observation but different consequences for the next action (Fig. 1). We study *hidden-mechanism interaction*: goal-directed tasks in which an agent must infer these states from observation–action history and decide which mechanism to activate, in what order, and for how long.

World models learn action-conditioned dynamics for control (Hafner et al., 2025), while visuomotor policies generate action sequences from observations (Chi et al., 2025). Executable representations also support robot policies (Liang et al., 2023), learned world models (Tang et al., 2024; Piriyakulkij et al., 2026), and planning under partial observability (Curtis et al., 2025). These advances motivate a specific design question: how can local transition structure guide both planning under uncertain state and revision of an individual mechanism? We address this question through belief-conditioned effect–precondition composition and evidence-validated local rule updates.

Our organizing principle is to separate a mechanism's transition rule, its current state, and uncertainty about that state. A lamp can occupy a different mode under the same cycle rule; changing the cycle order instead changes the rule. A *local causal program* specifies preconditions and intervention effects on a typed subset of state, with editable parameters such as a hold duration or threshold. We use parameterized hybrid templates for modes, latches, accumulators, and timed events, reflecting the combination of discrete control logic and continuous processes studied in hybrid systems (Henzinger, 1996; Fox & Long, 2006). The same executable rules support inference, composition, and revision.

We instantiate this principle in **CausalWAM**. A recurrent model infers a mixed discrete–continuous belief from history and retrieved program hypotheses. The planner binds programs to controls and composes chains whose predicted effects satisfy later preconditions under the evolving belief. A jointly trained action decoder grounds the selected operation into action chunks, with future-latent

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

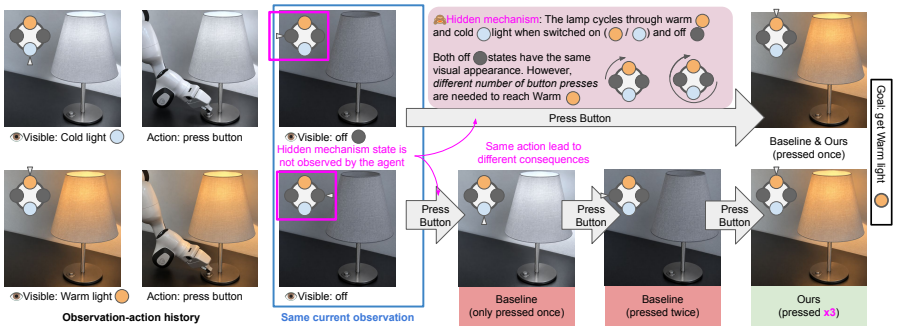

Figure 1: **Hidden-mechanism ambiguity.** Two interaction histories produce the same visible lamp's off state but different hidden cycle states. Pressing the same button then produces different outcomes, requiring the agent to retain action-relevant history. On row two, baselines stop prematurely or fail.

prediction providing dynamics supervision. After surprising outcomes, the update procedure first revises the state belief. When the current rule remains inadequate, it validates candidate edits against subsequent observations and stores accepted changes at the affected device binding, retaining shared program schemas for reuse.

We introduce **MechanismBench** to measure these capabilities through instrumented hidden state, alternate-action queries, held-out compositions, and local rule changes. Mechanism and program annotations provide training supervision; deployment uses observations, action history, goals, and internal memory. CausalWAM reaches 88.9% in-distribution success and 76.2% success on held-out compositions, and recovers to 85.4% success by the tenth post-change episode. It outperforms supervision-matched, programmatic, and multimodal-agent baselines on composition and adaptation. Controlled studies examine planning at matched search budgets, the effect of executable-rule edits on decisions, and selective repair under shared evidence. Franka robot experiments demonstrate physical execution on hidden-mode, accumulator, and delayed-trigger fixtures.

Contributions.

- **Mechanism-first formulation and abstraction.** We model hidden-mechanism interaction with local executable causal programs, separating transition rules, current state, and uncertainty about that state via a shared interface for prediction, composition, and revision.
- **Belief-conditioned program planning.** We introduce a causal world-action model that infers mechanism belief, constructs program chains through belief-conditioned effect–precondition matching, and grounds selected operations into learned action chunks.
- **Reusable program memory with selective repair.** We develop a library that retrieves local rules across tasks, validates proposed changes against fresh observations, and stores accepted repairs at affected device bindings while retaining shared schemas.
- **MechanismBench and physical validation.** We introduce a benchmark with evaluator-only state, intervention queries, held-out compositions, and controlled rule changes, complemented by instrumented robot experiments.

2 Related Work

Executable models and structured planning. Executable world models turn interaction experience into revisable dynamics for planning (Tang et al., 2024; Piriyakulkij et al., 2026), including probabilistic models for reasoning under partial observability (Curtis et al., 2025; Six et al., 2026). Learned relational operators expose preconditions and effects that connect transition structure to grounded actions (Chitnis et al., 2022). CausalWAM uses local causal hybrid programs to propagate hidden-state beliefs and construct invocation chains through effect–precondition matching. The distinction lies in how executable structure organizes computation, which we examine through matched-budget planning and direct interventions on program rules.

Embodied reasoning and program reuse. Grounded language models and generated policies connect perceptual inputs to robot decisions and reactive control (Driess et al., 2023; Liang et al., 2023). Program-library learning and continual skill acquisition support reusable abstractions across

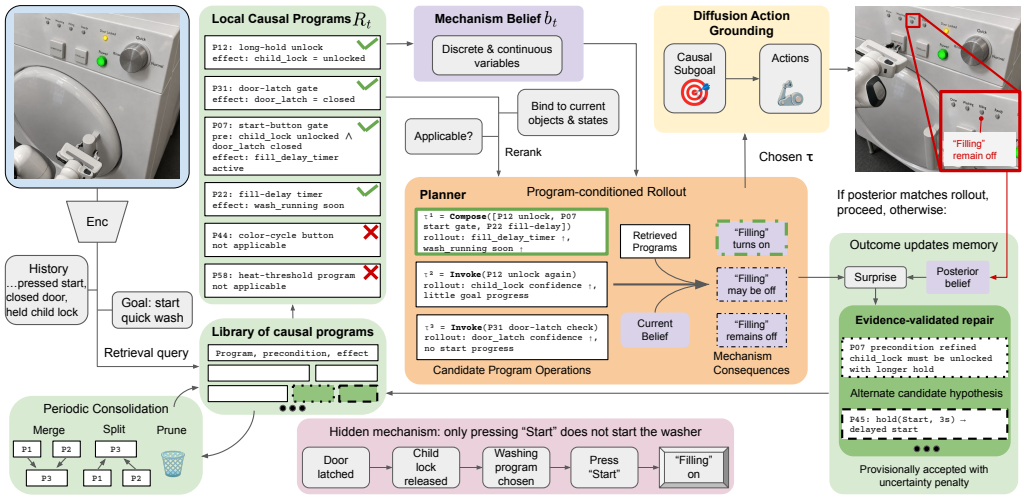

Figure 2: **CausalWAM inference.** History and retrieved programs determine a belief over hidden mechanism state. The planner executes candidate program chains internally and grounds the selected operation into actions. Observed outcomes update the belief and, when validated, the affected program instance. Program IDs displayed are local to this example.

problems (Ellis et al., 2021; Wang et al., 2024). Our library stores environment-transition rules used by inference and planning. Evidence-validated updates attach repairs to particular device bindings, allowing local mechanisms to change while preserving shared schemas.

Causal representations and world-action learning. Temporal and interventional learning can recover causal factors and transferable dynamics abstractions (Lippe et al., 2022; Wang et al., 2022). Complementary work couples action generation to future prediction (Shen et al., 2025) and conditions generated futures on intermediate motion and geometry (Xu et al., 2026). We use future-latent prediction to supervise action grounding, while executable hidden-state tarnsitions provide the interface for planning and mechanism repair.

Mechanism evaluation. Manipulation benchmarks study transfer across tasks and controlled physical variations (Ahmed et al., 2021; Liu et al., 2023). MechanismBench complements these evaluations with hidden transition programs, held-out chains, and local rule changes. Evaluator-only state and intervention queries distinguish state inference, composition, and repair; instrumented fixtures test physical execution.

3 Method

CausalWAM uses executable local programs as a shared interface for hidden-state inference, planning, and mechanism revision. Fig. 2 shows the inference loop: infer state under retrieved program hypotheses, compose their effects toward the goal, ground the next operation into actions, and update the relevant program when new evidence supports a rule change.

3.1 Problem Setup and Program Representation

The agent receives observations o_t and a goal g, and chooses actions from history $h_t = (o_{\leq t}, a_{<t})$. Action records include duration so that waiting and sustained input can be distinguished. We separate the hidden state $m_t = (d_t, c_t)$, containing discrete and continuous variables, from the rules governing its evolution. A local causal program is $\mathcal{P}_r = (\mathcal{V}_r, \mathcal{U}_r, \mathrm{Pre}_r, f_{\phi_r}, \Omega_r, \rho_r)$. Here $\mathcal{V}_r$ specifies typed variables, $\mathcal{U}_r$ intervention hooks, Pre_r preconditions, and f_{ϕ_r} a transition rule with editable parameters. The map Ω_r describes observability, while ρ_r records confidence, supporting evidence, and counterexamples. The rule exposes a typed effect signature and maps the scoped state m^r and hook input u^r to a distribution over the variables it writes. Variables outside its write scope are unchanged by that local transition.

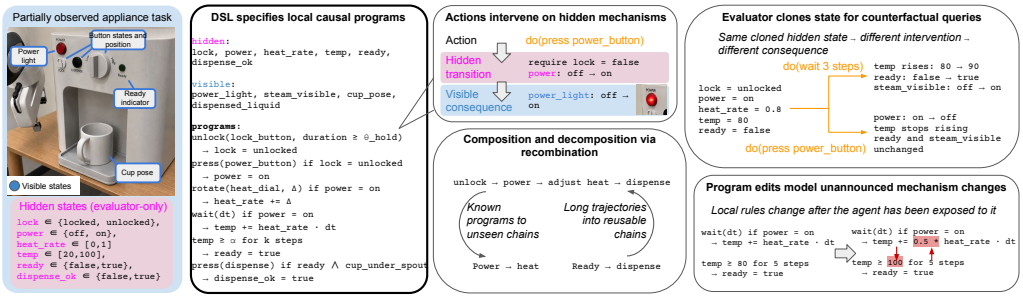

Figure 3: **MechanismBench.** Local programs define hidden dynamics and partial observations. Evaluator-only state supports intervention queries and controlled changes to composition, appearance, and transition rules. During interaction, the agent receives observations, action history, and the goal.

Templates specify toggles/cycles, latches/gates, counters, rate control, accumulators, thresholds, timers, stochastic gates, and preconditioned affordances. This vocabulary captures discrete control logic coupled to timed or continuous processes. The template forms are supplied; their selection, scene bindings, and parameter values are learned from interactions. Neural proposal and binding modules instantiate these forms, with template parameters and local transition heads specifying quantitative dynamics. For example, an unlock program maps $\mathrm{hold}(\mathrm{button}, \eta)$ to $\mathrm{lock} = \mathrm{unlocked}$ when $\eta \geq \theta_{\mathrm{hold}}$. The duration η is chosen by the agent; θ_{hold} belongs to the transition rule. An alternate-action query changes the hook input while holding the rules fixed; a model repair revises the estimated transition rule. Appendix B detail the template semantics.

3.2 Inferring Mechanism Belief from History

Aliased observations require retaining which actions occurred and which effects followed. Image features from a pretrained visual encoder, together with available proprioceptive and visible symbolic inputs, pass through a Perceiver-style bottleneck to produce fixed-size perceptual tokens x_t (Jaegle et al., 2021). A recurrent state summarizes history, following recurrent state-space modeling (Hafner et al., 2019); typed heads produce categorical probabilities for discrete variables and Gaussian parameters for continuous variables.

Let b_t^- denote the predictive state belief before observing the latest outcome, obtained from the previous belief and executed action. Retrieval uses this prediction together with the observed history, and the posterior then incorporates the retrieved programs: $R_t = \mathrm{Retrieve}_\theta(\mathcal{L}, h_t, x_t, b_t^-, g)$, wherr $b_t = q_\theta(m_t \mid h_t, R_t)$. At initialization, b_0^- is the initial state prior. This ordering lets program hypotheses explain evidence such as a long hold with no visible response. The belief represents uncertainty about mechanism state; evidence metadata in ρ_r separately tracks confidence in the retrieved transition rules.

3.3 Retrieving and Updating Local Programs

Reuse requires identifying applicable transition knowledge across different appearances and control bindings. The library $\mathcal{L}$ stores program schemas, binding-specific instances, learned retrieval keys, and evidence metadata. Vector search returns 16 candidates; structured reranking retains up to five using type compatibility, hook availability, goal relevance, precondition likelihood, and confidence. Precondition likelihood is a soft ranking feature: programs remain candidates when their preconditions can be enabled by earlier operations. Their preconditions are checked under the predicted belief at the point of invocation.

Evidence-validated repair. A surprising outcome can arise from uncertainty about state. We first update the state belief under the existing program, using observation-latent prediction error and predictive–posterior disagreement to identify unexplained transitions. The unchanged rule with its updated state belief remains the reference explanation when evaluating a repair. Persistent unexplained outcomes, event boundaries, or retrieval failure trigger template-guided proposals from a trailing interaction window. Candidates modify parameters, bindings, or preconditions, or replace a local template with a compatible alternative.

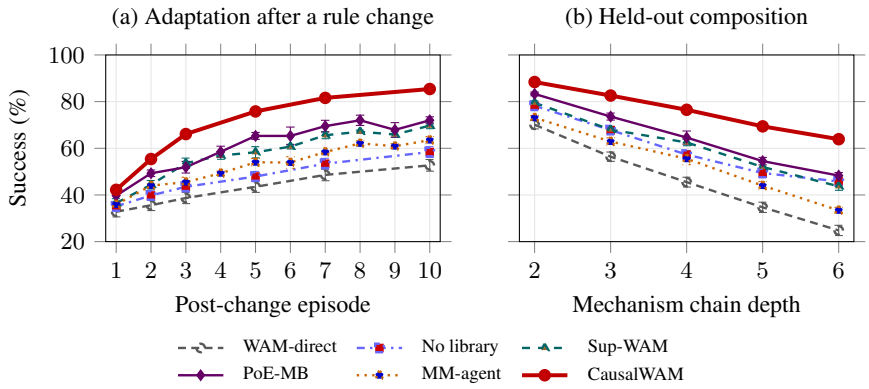

Figure 4: **Program memory supports recovery and compositional reuse.** CausalWAM recovers more rapidly after unannounced rule changes (a) and maintains higher success as held-out chains lengthen (b). The no-library comparison supports the contribution of reusable, editable programs to both capabilities. Lines connect displayed points; adaptation AUC uses all ten post-change episodes. Error bars are SEM across five seeds.

A neural proposal head scores templates from the diagnostic window, belief changes, recent actions, and retrieved entries; fitting estimates the candidate parameters within the selected template. Candidates are then validated on subsequently observed transitions, separate from the fitting evidence. For candidate edit e, let $\mathcal{P}_r \oplus e$ denote the modified rule. We score $\Delta(e) = L_{\text{obs}}(\mathcal{P}_r; S_{\text{val}}) - L_{\text{obs}}(\mathcal{P}_r \oplus e; S_{\text{val}}) - \lambda_e \,\text{size}(e)$, where S_{val} contains fresh observed transitions, L_{obs} scores observation-latent and visible-event predictions, and $\text{size}(e)$ penalizes changed fields and expanded scope. Both hypotheses filter the same history and predict each validation outcome before observing it; candidate parameters remain fixed during validation. An edit must pass type and binding checks, exceed an improvement threshold,and accumulate supporting evidence. Thresholds are chosen on development sessions, including state changes and observation disturbances with unchanged rules. Online validation uses agent observations, not evaluator states or cloned outcomes.

Accepted changes are stored as versioned overrides indexed by program schema and device binding. Retrieval resolves an override only for its binding, preserving the shared schema for other instances. Validated entries enter as provisional programs with uncertainty penalties; further support and a low counterexample rate permit promotion. Periodic consolidation merges, splits, abstracts, or prunes entries outside the control loop. Local repair supports short-horizon adaptation; consolidation manages longer-term reuse. Appendix C details proposal, validation, and library maintenance.

3.4 Planning in Mechanism Space

To reach a goal, the agent must determine which local effects enable subsequent actions. The planner selects an operation $\tau_t = (\text{op}, \mathcal{C}, \beta, \eta)$, where $\mathcal{C}$ is a program or short chain, β binds variables and hooks to the current scene, and η specifies intervention parameters. Bind instantiates a schema, Invoke calls its hook, and Compose links typed effects to later preconditions. Test denotes internal prediction; program edits belong to library maintenance.

A small transformer attends to belief, goal, and program tokens, with attention biases reflecting variable, hook, and precondition–effect connections. It proposes K candidate operations, constructing chains through compatible bindings and effect–precondition links. Each extension is evaluated at the predicted intermediate state, so a program can become applicable after a preceding effect. The executor propagates the belief through each bound chain: $\hat{b}^{(k)}_{t+1:t+H_m} = \text{Exec}_\theta(b_t, R_t, \tau_t^{(k)})$. Program tokens guide proposal, while the bound transition rules and their parameters determine the internal rollout. During an intervention's elapsed time, active accumulators and timers evolve and guarded events fire according to their templates. A readiness threshold is therefore a predicted event, not a state assignment that the planner can command directly.

Candidate scores combine predicted goal progress, action cost, uncertainty, and precondition violations: $S_\theta(\tau) = G_\theta(\hat{b}, g) - \lambda_c C_\theta(\tau) - \lambda_u U(b_t, R_t, \tau) - \lambda_p \,\text{Viol}(\hat{b}, \tau)$. The uncertainty term accounts for belief, program confidence, and binding ambiguity. The violation term sums the probabilities

that preconditions fail at their corresponding intermediate beliefs. The highest-scoring candidate is selected as $\tau_t^\star$. An unlock effect can thus enable a power operation, which enables accumulation and eventually a readiness-dependent affordance. The library stores the local rules; the planner constructs their chains and reevaluates them as observations arrive. Defaults are $K = 8$, maximum chain length four, and mechanism rollout horizon $H_m = 5$, with replanning for longer tasks.

3.5 Grounding Operations into Actions

An intended mechanism change must be realize through the available controls. A conditional diffusion decoder (Chi et al., 2025) generates action chunks of horizon H $a_{t:t+H-1} \sim \pi_\theta(\cdot \mid b_t, x_t, R_t, g, \tau_t^\star)$. For a composed target, the decoder realizes the first executable intervention in the chain. The agent executes a short prefix, incorporates the observed outcome, and replans. Conditioning on the selected operation separates choosing the next mechanism change from realizing it through motion. A future-latent head jointly predicts observation embeddings and mechanism consequences, coupling action grounding to dynamics during training. Online program rollouts operate in mechanism space; observation-latent prediction also supplies evidence for repair.

3.6 Training Objective

We pretrain on trajectories, bootstrap the library, and jointly fine-tune the neural modules using losses for typed-state inference, action-conditioned dynamics, program induction, operation selection, action denoising, future-latent prediction, and retrieval. MechanismBench supplies training annotations for hidden state, templates, bindings, parameters, and successful chains, together with intervention outcomes. Retrieval and library writes remain discrete. At deployment, neural weights are fixed and observations update beliefs and program entries. Appendix E specifies further training details.

3.7 MechanismBench

A task DSL specifies entities, hooks, local transition programs, observability masks, and outcome goals. The generator composes programs from the same template vocabulary used by the model, varying their bindings, parameters, and visual rendering. This evaluates reuse within the specified mechanism abstraction. Test splits isolate observation aliasing, held-out compositions and subchains, nuisance appearance changes, and unannounced local rule changes.

The evaluator logs hidden state and clones it for alternate-action queries; the agent receives the candidate action and query specification, but not the cloned state or outcome. A specified hypothetical edit is supplied as part of its query; an unannounced rule change must instead be inferred from subsequent interaction. Diagnostic queries are read-only and cannot update agent memory. Goals use outcome predicates encoded as tokens, with the same goal representation across methods. During control and unannounced-change adaptation, task programs, hidden state, and edit annotations remain evaluator-side. Appendix D specifies the DSL, query interfaces, and split construction.

4 Results

We evaluate whether executable mechanism programs support hidden-state inference, compositional control, and adaptation, and whether these capabilities carry through to physical interaction.

Experimental setup. MechanismBench uses 12,000 training and 2,000 validation episodes, with held-out tests for observation aliasing, composition, nuisance shifts, and rule changes. Agents receive task observations, previous actions, elapsed time, and goals; hidden state and edit annotations remain evaluator-side. We compare latent WM+MPC, which searches learned latent dynamics Hafner et al. (2019); WAM-direct, which jointly predicts actions and future latents Shen et al. (2025); a supervision-matched WAM (Sup-WAM); an adaptation of PoE-World (PoE-MB) (Piriyakulkij et al., 2026); and a fixed multimodal agent (MM-agent) (ChatGPT Sol 5.6) OpenAI (2026). Sup-WAM matches the encoder, action decoder, training trajectories, and auxiliary mechanism annotations, but uses auxiliary prediction heads instead of executable program planning. PoE-MB and MM-agent retain interaction evidence through their respective program or context updates. A state-assisted reference additionally receives the current hidden state.

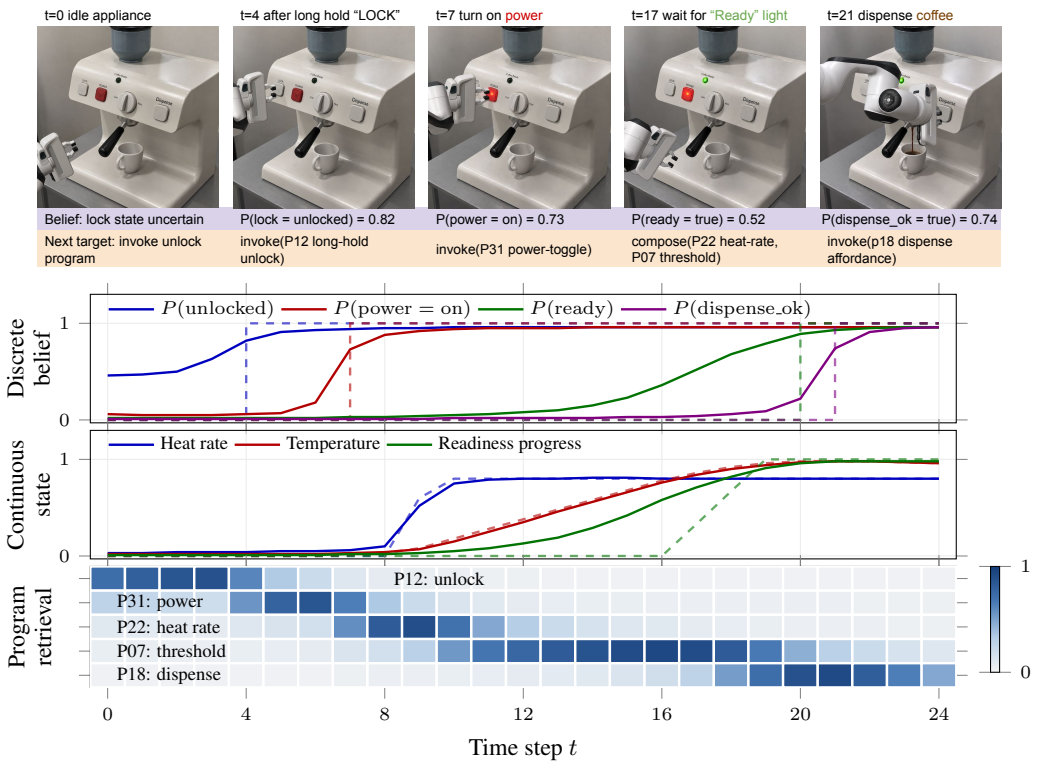

Figure 5: **Belief and retrieval follow successive mechanism stages.** Retrieved programs shift from unlocking and power control to readiness and dispensing as the inferred state evolves. The trace illustrates how local programs organize a multistage interaction with delayed visible effects. Solid curves show model beliefs and normalized continuous estimates; dashed curves show evaluator-only state. The heatmap reports program retrieval scores.



Figure 6: **Physical mechanism interaction.** Instrumented hidden-mode, accumulator-threshold, and delayed-trigger fixtures require state-dependent action selection and timing. Keyframes show examples of CausalWAM interactions and baseline failures, with further illustrations of the mechanism on the left. Hidden fixture logs are used only for evaluation and resets.

Table 1: **MechanismBench capabilities.** ID denotes in-distribution task success; prediction accuracy concerns specified alternate-action queries. All metrics use a 0–100 scale; higher is better. The state-assisted reference receives privileged current state.

Method	ID success	Prediction	Composition	Adaptation AUC
Latent WM+MPC Hafner et al. (2019)	70.8 ± 1.7	65.4 ± 1.3	42.5 ± 2.1	46.5 ± 1.8
WAM-direct Shen et al. (2025)	75.2 ± 1.5	71.2 ± 1.0	46.3 ± 1.9	43.9 ± 1.9
Sup-WAM	84.2 ± 1.0	78.2 ± 0.6	61.1 ± 1.0	57.9 ± 0.5
PoE-MB Piriyakulkij et al. (2026)	84.1 ± 0.8	80.9 ± 0.7	64.9 ± 1.1	61.1 ± 1.9
MM-agent OpenAI (2026)	78.0 ± 1.1	77.0 ± 1.0	53.8 ± 1.3	52.7 ± 0.5
CausalWAM	$\mathbf{88.9 \pm 0.8}$	$\mathbf{88.3 \pm 0.6}$	$\mathbf{76.2 \pm 1.2}$	$\mathbf{73.0 \pm 1.1}$
State-assisted	93.1 ± 0.5	94.0 ± 0.4	83.5 ± 0.9	86.5 ± 0.7

Task success measures goal completion within the action budget. Alternate-action accuracy uses a common held-out history bank and specifies the candidate action, horizon, and continuation; evaluator rollouts supply labels without updating agent memory. Composition success averages equally over held-out depths two through six. Adaptation AUC is mean success across ten post-change episodes on a 0–100 scale. We report means and standard errors over five seeds. Appendix E specifies supervision, interfaces, diagnostic probes, and baseline configurations.

Can the model infer what it cannot see? Observation-aliasing episodes contain similar observations with different action consequences. CausalWAM improves alternate-action prediction (Tab. 1) and hidden-state recovery: discrete-state F1 rises from 73.6% for WAM-direct to 88.8%, while continuous NRMSE falls from 0.163 to 0.076. Figure 5 illustrates how belief and retrieval evolve through a multistage apliance interaction. Together, these results support a belief representation that retains action-relevant distinctions across aliased histories.

Does planning over mechanisms improve control? CausalWAM achieves higher in-distribution success than the direct-action baselines, including Sup-WAM (Tab. 1). Removing the planner reduces success despite retaining mechanism belief and retrieved programs (Tab. 2a). Together, these comparisons support selecting mechanism-level operations as an effective step between state inference and action grounding.

Can local programs solve new mechanism chains? Held-out compositions recombine familiar local programs into unseen complete chains. CausalWAM leads aggregate success and retains its advantage as chains lengthen (Tab. 1; Figure 4b). Removing the library or replacing programs with raw episodes reduces composition success (Tab. 2a). These results support storing reusable local transitions and constructing their chains during planning.

Can the model recover when a mechanism changes? Each session introduces an unannounced local rule change followed by ten episodes. Environment state resets between episodes, while permitted agent memory persists and neural weights remain fixed; independent sessions restart from the trained model and memory. CausalWAM recovers faster than the compared baselines, and its advantage grows with experience (Figure 4a). Removing program memory or replacing it with raw episodes slow recoverys (Tab. 2a). Editable local rules thus support restoring control from post-change evidence.

4.1 Which Design Choices Explain the Gains?

Tab. 2a shows complementary benefits from future co-training, program planning, and memory. Three controlled comparisons examine how the program interface contributes.

Does structure improve search? Typed-chain and flat primitive search share beliefs, programs, execution, and scoring. At a matched budget of primitive transition evaluations per decision, typed search achieves higher held-out composition success. Matching rollout computation supports an advantage from organizing search through effect–precondition links.

Do executable rules guide decisions? With observations, belief, goal, and weights fixed, updating a rule and its tokens improves edited-outcome predictions and operation choices over changing tokens alone. Agreement on unaffected queries remains high. This selective response supports execution of the rule as a functional part of decision making.

Table 2: **Design choices and selective repair.** (a) Architectural ablations on the main splits. (b) Controlled repair after ten shared exposure episodes; these are separate from closed-loop adaptation. Param. and template columns give changed-device success (%); false commits measure rule edits under state-only changes or observation disturbances (%). Δ unaffected is the paired change in unaffected-device success, in percentage points.

(a) Architectural ablations

Variant	ID success ↑	Composition ↑	Adaptation AUC ↑
CausalWAM	$\mathbf{88.9 \pm 0.8}$	$\mathbf{76.2 \pm 1.2}$	$\mathbf{73.0 \pm 1.1}$
No future co-training	84.1 ± 1.0	70.5 ± 1.3	68.4 ± 1.2
No program planner	80.6 ± 1.2	62.8 ± 1.5	66.0 ± 1.3
No program library	84.7 ± 1.0	59.7 ± 1.6	48.7 ± 1.8
Raw episodic retrieval	86.0 ± 0.9	63.4 ± 1.5	59.1 ± 1.5

(b) Controlled repair

Update variant	Param. ↑	Template ↑	False commits ↓	Δ unaffected
Belief update only	63.0 ± 2.6	52.3 ± 2.0	0.0 ± 0.0	0.00 ± 0.00
Validated parameter edits	84.4 ± 0.6	58.0 ± 2.3	2.8 ± 0.7	-0.70 ± 0.18
No edit validation	81.5 ± 0.9	69.4 ± 1.7	15.4 ± 1.2	-3.30 ± 0.28
Shared-schema overwrite	91.0 ± 0.2	82.0 ± 1.3	3.2 ± 0.2	-12.68 ± 0.63
Binding-scoped repair	91.0 ± 0.2	82.0 ± 1.3	3.2 ± 0.2	-0.30 ± 0.13

Table 3: **Physical fixture success**. Entries give successes out of 60 trials, followed by success percentage and SEM across three evaluation blocks of 20 trials.

Method	Hidden mode	Accumulator	Delayed trigger
WAM-direct Shen et al. (2025)	$35/60$ (58.3 ± 6.0)	$36/60$ (60.0 ± 7.6)	$31/60$ (51.7 ± 6.0)
Sup-WAM	$45/60$ (75.0 ± 7.6)	$43/60$ (71.7 ± 4.4)	$35/60$ (58.3 ± 3.3)
PoE-MB Piriyakulkij et al. (2026)	$44/60$ (73.3 ± 1.7)	$42/60$ (70.0 ± 2.9)	$32/60$ (53.3 ± 4.4)
MM-agent OpenAI (2026)	$42/60$ (70.0 ± 5.0)	$38/60$ (63.3 ± 6.0)	$26/60$ (43.3 ± 1.7)
CausalWAM	$\mathbf{50/60}$ $\mathbf{(83.3 \pm 6.0)}$	$\mathbf{50/60}$ $\mathbf{(83.3 \pm 8.8)}$	$\mathbf{50/60}$ $\mathbf{(83.3 \pm 4.4)}$

Can repairs stay local? Variants receive ten shared diagnostic episodes followed by read-only evaluation, separating updating from evidence gathering. Two devices share a schema, but only one device's rule changes; state-only changes and observation disturbances test false commits. Tab. 2b separates the benefits: full repairs better accommodate template changes, validation suppresses spurious edits, and binding-scoped writes limit disruption to the unaffected device. Together, these controls support selective revision of mechanism knowledge.

4.2 Does Mechanism Reasoning Transfer to a Robot in the Physical World?

A Franka operates hidden-mode, accumulator-threshold, and delayed-trigger fixtures using camera observations and proprioception; instrumentation supplies evaluator state and resets. CausalWAM succeeds more reliably across all three fixtures (Tab. 3; Figure 6). These results support grounding mechanism-dependent decisions into physical actions, including when success requires tracking hidden progress or waiting for delayed effects.

5 Conclusion

We presented CausalWAM, a mechanism-first world-action model that separates transition rules from hidden state and uncertainty about that state. Its executable programs support belief-conditioned composition and learned action grounding; evidence-validated, binding-scoped edits provide a mechanism for selective repair. MechanismBench measures these capabilities through hidden-state instrumentation, alternate-action queries, held-out compositions, and local rule changes. The evaluations support the value of structured planning and editable program memory for compositional control and adaptation, with instrumented Franka fixtures demonstrating physical execution. The hybrid template vocabulary provides a controlled basis for studying how mechanism knowledge is inferred, composed, and revised.

AI USE STATEMENT

In this work, we used generative AI tools for latex formatting bugs and identifying grammar problems only. We have not used generative AI tools for other tasks, and the rest of the required disclosure tasks are not applicable to this work. We have reviewed all AI-assisted work manually. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Incorrect mechanism estimates or program updates can lead to mistimed or unsafe actions. Electronic appliances can also themselves pose safety risks. Our physical experiments use human supervision, workspace and velocity limits, force/torque thresholds, and an emergency stop. Further deployment should include task-specific safety validation and retain safeguards independent of the learned mechanism model.

REPRODUCIBILITY STATEMENT

We will release the code and data used in producing the manuscript upon publication.

REFERENCES

Ossama Ahmed, Frederik Träuble, Anirudh Goyal, Alexander Neitz, Manuel Wüthrich, Yoshua Bengio, Bernhard Schölkopf, and Stefan Bauer. Causalworld: A robotic manipulation benchmark for causal structure and transfer learning. In *International Conference on Learning Representations*, 2021.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Rohan Chitnis, Tom Silver, Joshua B Tenenbaum, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Learning neuro-symbolic relational transition models for bilevel planning. In *2022 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 4166–4173. IEEE, 2022.

CODESYS GmbH. Templates, 2006. URL `https://content.helpme-codesys.com/en/LibDevSummary/templates.html`. CODESYS Library Development Guidelines.

Aidan Curtis, Hao Tang, Thiago Veloso, Kevin Ellis, Joshua B Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Llm-guided probabilistic program induction for pomdp model estimation. In *Conference on Robot Learning*, pp. 3137–3184. PMLR, 2025.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: an embodied multimodal language model. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. Dreamcoder: bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, pp. 835–850, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383912. doi: 10.1145/3453483.3454080. URL `https://doi.org/10.1145/3453483.3454080`.

Maria Fox and Derek Long. Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, 27:235–297, 2006.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pp. 2555–2565. PMLR, 2019.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, pp. 1–7, 2025.

T. A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, LICS '96, pp. 278, USA, 1996. IEEE Computer Society. ISBN 0818674636.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pp. 4651–4664. PMLR, 2021.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International conference on robotics and automation (ICRA)*, pp. 9493–9500. IEEE, 2023.

Phillip Lippe, Sara Magliacane, Sindy Löwe, Yuki M Asano, Taco Cohen, and Stratis Gavves. Citris: Causal identifiability from temporal intervened sequences. In *International Conference on Machine Learning*, pp. 13557–13603. PMLR, 2022.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 44776–44791. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/8c3c666820ea055a77726d66fc7d447f-Paper-Datasets_and_Benchmarks.pdf.

OpenAI. Chatgpt (5.6 sol). https://chatgpt.com, 2026. Accessed: 2026-09-23.

Top Piriyakulkij, Yichao Liang, Hao Tang, Adrian Weller, Marta Kryven, and Kevin Ellis. Poe-world: Compositional world modeling with products of programmatic experts. *Advances in Neural Information Processing Systems*, 38:26609–26638, 2026.

Yichao Shen, Fangyun Wei, Zhiying Du, Yaobo Liang, Yan Lu, Jiaolong Yang, Nanning Zheng, and Baining Guo. VideoVLA: Video generators can be generalizable robot manipulators. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=UPHlqbZFZB.

Valentin Six, Frederik Panse, Mathis Fajeau, Lancelot Da Costa, Mridul Sharma, Alfonso Amayuelas, Tim Z Xiao, David Hyland, Philipp Hennig, and Bernhard Schölkopf. Learning pomdp world models from observations with language-model priors. *arXiv preprint arXiv:2605.13740*, 2026.

Hao Tang, Darren Key, and Kevin Ellis. Worldcoder, a model-based llm agent: Building world models by writing code and interacting with the environment. *Advances in Neural Information Processing Systems*, 37:70148–70212, 2024.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a.

Zizhao Wang, Xuesu Xiao, Zifan Xu, Yuke Zhu, and Peter Stone. Causal dynamics learning for task-independent state abstraction. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 23151–23180. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/wang22ae.html.

Ziming Xu, Shuang Liang, Ruobing Han, Ziqiao Xi, Mingxing Rao, Kun Zhou, Zijun Zhang, Yuchen Yan, Yufan Wei, Junbo Huang, Yifei Shao, Fang Nan, and Biwei Huang. Causalwm: Causal chain-of-thought reasoning for embodied world model, 2026. URL https://arxiv.org/abs/2609.23184.

A Appendix

In thss appendix, we report further implementation-side details on local causal programs, modules of the proposed pipeline such as program induction and repair, the specification of the proposed MechanismBench, training and baselines, additional supporting results, and further details on the physical evaluations.

B Local Causal Program Representation

This section specifies the executable interface introduced in Section 3.1: how local rules read and write state, how time-dependent effects are represented, and how bound programs compose under uncertainty. The same interface is used for internal prediction and selective repair.

B.1 Programs, State, and Belief

A program specifies a transition rule; mechanism state records its current values; belief represents uncertainty about those values. We retain the main-text notation

$$\mathcal{P}_r = (\mathcal{V}_r, \mathcal{U}_r, \mathrm{Pre}_r, f_{\phi_r}, \Omega_r, \rho_r), \qquad m_t = (d_t, c_t), \qquad b_t = q_\theta(m_t \mid h_t, R_t). \tag{1}$$

Discrete variables include modes, latches, event flags, and integer counters; continuous variables include rates, accumulators, and elapsed or remaining time. Progress within a hold, persistence condition, or scheduled delay belongs to state. The required hold duration, threshold, or delay belongs to the rule parameters ϕ_r. An observation can therefore revise the inferred progress without requiring a program edit.

A schema uses typed variable and control placeholders; a binding β maps them to the current scene and mechanism-variable slots. For an unlock program, the planner chooses a hold duration η, while the rule stores the required duration θ_{hold}. Changing η changes the attempted intervention; changing θ_{hold} revises the model of the device. Shared neural parameters are denoted by θ and remain fixed during online adaptation. Editable program parameters are stored explicitly in library entries.

B.2 Scope, Effects, and Observations

Read and write scope. Each variable declaration records its type, domain, and read/write role. Let $\mathcal{W}_r \subseteq \mathcal{V}_r$ be the write scope. The hook set $\mathcal{U}_r$ specifies action families and their arguments, such as a button press, a sustained hold, or a dial rotation. A bound precondition is a predicate on the scoped state and hook arguments. It is Boolean for a particular state, but has a satisfaction probability under a belief. Type compatibility and control availability are checked separately from uncertain state preconditions.

For a matching hook u, the local transition has the semantics

$$K_{r,\beta}(m' \mid m, u) = \begin{cases} f_{\phi_r}(m'_{\mathcal{W}_r} \mid m_{\mathcal{V}_r}, u)\, \delta_{m_{\overline{\mathcal{W}}_r}}(m'_{\overline{\mathcal{W}}_r}), & \mathrm{Pre}_r^{\beta}(m, u), \\ \delta_m(m'), & \text{otherwise.} \end{cases} \tag{2}$$

Scopes in this expression are resolved through β, and δ denotes a point mass. The identity branch means that this hook produces no program effect; other active processes can still evolve while time elapses. The effect signature exposes which variables can change and the relevant predicates or continuous changes. It guides chain construction, while execution determines whether the effect occurs from the current state. Neural transition heads, where used, predict within the declared types and write scope.

Observability and evidence. The field Ω_r links mechanism state to visible, delayed, or indirect observations. A hidden variable can affect an observable indicator without being directly supplied to the agent. The evaluator uses the task observation map; the agent uses observation predictions learned from training and its inferred programs. Metadata ρ_r records supporting segments, predictive error, counterexamples, confidence, and usage. Program confidence is distinct from the state belief and is used as a planning penalty rather than treated as a calibrated posterior over all possible programs.

B.3 Hybrid Templates and Time-Dependent Execution

Why this template family? The templates reflect recurring structure in engineered control systems. Hybrid automata describe discrete mode changes coupled to continuous evolution (Henzinger, 1996), while mixed discrete–continuous planning distinguishes commanded actions, ongoing processes, and condition-triggered events (Fox & Long, 2006). Bistable elements, counters, timers, and edge triggers also appear as standard controller building blocks such as CODESYS (CODESYS GmbH, 2006). These distinctions align with the decisions studied here: whether a control is enabled, how repeated or sustained input changes progress, and when a delayed effect becomes available. Parameterization and binding turn each transition form into a family of device-specific mechanisms; composition connects their effects into longer tasks. The vocabulary therefore supplies a structural prior while leaving mechanism state, control bindings, quantitative parameters, and local rule changes to be inferred from interaction.

The nine templates below expose discrete control logic and timed processes through a common interface. Each instantiated rule specifies its variable domains, units, boundary behavior, and reset conditions. The displayed rules are representative instances; sampled parameters and bindings distinguish individual mechanisms.

Toggle and cycle. A matching invocation applies a discrete transition map $d' = \sigma_r(d)$. A toggle is the two-state case; cycle order and control binding are editable. For Figure 1, the lamp uses four hidden states:

$$\text{warm} \to \text{off}_{\text{after warm}} \to \text{cold} \to \text{off}_{\text{after cold}} \to \text{warm}. \tag{3}$$

The observation map renders both off states identically, although their next outputs differ.

Latch and gate. A successful intervention sets a persistent condition that enables another program. For an uninterrupted hold of duration η, a latch unlocks when $\eta \geq \theta_{\text{hold}}$. When the hold spans action chunks, its accumulated duration remains in state and the guard is evaluated on that accumulated duration. Release and reset behavior belong to the instantiated rule.

Counter. A qualifying event increments an integer state, and a guard triggers an effect at the required count. The event definition, required count, and reset condition determine the instance. Repeated observations of the same event do not constitute additional increments.

Rate control and accumulation. A rate-control program maps intervention magnitude to a rate change; an additive instance is $v' = \text{clip}(v + \kappa u)$. An accumulator advances a continuous state under an active condition, for example

$$c' = \text{clip}(c + \gamma v \Delta t), \tag{4}$$

where γ converts the rate and elapsed time to state units. Gain, decay, and saturation behavior belong to the instantiated rule. Elapsed time includes time spent executing other actions, not only explicit waiting.

Threshold and persistence. A threshold maps a continuous condition to a discrete event. Persistence requires additional state: for a condition checked at declared mechanism ticks,

$$n_{s+1} = \begin{cases} \min(n_s + 1, k), & c_{s+1} \geq \alpha, \\ 0, & c_{s+1} < \alpha, \end{cases} \qquad \text{trigger}_{s+1} = \mathbf{1}[n_{s+1} = k]. \tag{5}$$

Here α and k are rule parameters, while n_s is hidden progress. A latched readiness effect persists until its specified reset. Mechanism ticks are part of the transition semantics and need not coincide with planner decisions.

Delay and timer. An invocation arms a pending effect and initializes timer state. For a countdown instance, remaining time evolves as $q' = \max(0, q - \Delta t)$, and the effect fires when an active timer expires. The delay parameter, retrigger behavior, cancellation condition, and post-fire reset specify the instance. An inactive timer and an expired active timer are distinct states.

Stochastic gate. An enabled invocation produces its effect according to a probability stored in the rule, such as $z \sim \text{Bernoulli}(p_r)$. Predictions retain this outcome distribution; an individual unsuccessful invocation is not automatically a counterexample to the mechanism.

Preconditioned affordance. An action produces its effect only when a conjunction of hidden and observed conditions holds. For example, dispensing requires readiness and a cup under the spout. The control binding, condition structure, and effect are part of the program instance.

Execution over elapsed time. A rollout combines commanded hooks, active continuous processes, and guarded or scheduled events. The executor advances processes over the supplied durations and applies the declared guards and event semantics. Simultaneous writes require an explicit order or joint transition rule; the retrieval ranking does not define that order. Internal clocks and persistence progress remain in state across action-chunk boundaries. The numerical stepping and event conventions are part of the task/template specification, shared by control rollouts and diagnostic queries.

B.4 Operations and Belief-Conditioned Composition

The target operation is $\tau = (\mathrm{op}, \mathcal{C}, \beta, \eta)$. BIND instantiates a schema, INVOKE selects its intervention hook, and COMPOSE links bound programs through effect–precondition compatibility. The parameters η specify actions or desired effects, such as a duration, rotation, or target mode; selecting a target does not assign that hidden state directly. TEST executes an internal prediction query. EDIT changes a stored model during library maintenance, rather than acting on the physical device.

For a resolved bound chain, let $T_{\mathcal{C},\beta,\eta}$ denote its hybrid transition kernel, including autonomous evolution during each intervention. Its predictive belief is

$$\hat{b}'(m') = \int T_{\mathcal{C},\beta,\eta}(m' \mid m) b(m) \, \mathrm{d}m, \tag{6}$$

with summation over discrete state components. This defines the prediction represented by the learned typed belief and transition heads; it does not require exact integration in the implementation. Each precondition is evaluated at its corresponding intermediate belief, rather than only at the chain's starting state. Alternative hypotheses for the same mechanism are evaluated as alternative rules, not simultaneous writes to one state variable.

B.5 Example: From Unlocking to Dispensing

The appliance example composes an unlock program, a power toggle, rate control, accumulation, readiness, and a dispensing affordance. The unlock effect enables power control; power and the selected rate enable accumulation; threshold persistence produces readiness; readiness and cup placement enable dispensing. Waiting is an intervention that allows active processes to evolve, whereas readiness is a predicted event. The planner grounds the next executable intervention, observes the outcome, and replans rather than issuing the entire dependency chain open-loop.

An alternate-action query holds the current belief and rules fixed while changing the next hook and its arguments. A hypothetical rule edit changes a copy of one program and predicts its consequences. An unannounced physical rule is change will instead requires the agent to revise its model from subsequent observations. These uses share the execution interface but have different information contracts, specified in Appendix D.

C Program Induction, Retrieval, and Selective Repair

This section expands Section 3.3 and provides the online algorithm moved from the main text. The library supplies reusable transition hypotheses; state inference explains current observations under those rules, and fresh predictive evidence determines whether a local rule should be revised.

C.1 Library Entries and Program Lifecycle

The library contains reusable schemas and binding-specific instances. An instance records its schema identifier, scene/control binding, version, local parameter or structural changes, and evidence metadata. Binding identifiers refer to the agent's scene associations, not evaluator program labels. For a particular binding, retrieval resolves the active accepted instance; otherwise it uses the shared schema.

A *candidate* is a proposed explanation held outside planning memory. A *provisional* entry has passed structural checks and fresh-evidence validation, and can be retrieved with an uncertainty penalty. A *trusted* entry has accumulated additional support with a sufficiently low counterexample rate. Each version retains its evidence provenance, so subsequent failures can reduce confidence without changing unrelated entries.

C.2 Retrieval, Binding, and Planning

Retrieval begins from the predictive belief b_t^-, current perceptual tokens, goal, and available history. Approximate vector search returns up to $K_R = 16$ candidates. After resolving instance versions, a learned binding module matches schema variables and hooks to the current context. Candidates with incompatible types or unavailable hooks are excluded; state preconditions contribute a soft score:

$$\begin{aligned} s(r, \beta) = s_{\text{vec}}(r) + \lambda_{\text{pre}} P_{b_t^-}(\text{Pre}_r^\beta) + \lambda_{\text{eff}} \text{Rel}(\text{Eff}_r^\beta, g) \\ - \lambda_{\text{unc}} \text{Unc}(\rho_r) - \lambda_{\text{bind}} \text{Cost}(\beta). \end{aligned} \tag{7}$$

The top $K_R' = 5$ bound entries form R_t, or fewer when fewer valid bindings exist. A false current precondition does not eliminate a program whose precondition may be enabled by a preceding operation. The posterior $b_t = q_\theta(m_t \mid h_t, R_t)$ is inferred after retrieval. This ordering avoids querying with a posterior that already depends on the retrieved set.

The planner proposes $K = 8$ operations with maximum chain length $D = 4$ and mechanism horizon $H_m = 5$. It executes and scores them as in Section 3.4. Effect–precondition matching proposes chains; intermediate belief propagation determines their feasibility and progress. Task depth and planning depth are distinct: replanning can solve a six-transition task through shorter candidate chains. The library therefore stores local rules rather than every complete task sequence. Pending effects remain part of the modeled state as time advances, rather than restarting at each retrieval.

C.3 State Inference before Rule Proposal

Before proposing a repair, the agent updates state under the unchanged rule. At transition $t \to t+1$, it compares the cached prediction with the new observation and posterior:

$$\epsilon_t = \lambda_x \ell_x(x_{t+1}, \hat{x}_{t+1|t}) + \lambda_y \ell_y(y_{t+1}, \hat{p}_{t+1|t}) + \lambda_b D_{\text{KL}}(b_{t+1}^{\text{post}} \| b_{t+1}^-). \tag{8}$$

Here y_{t+1} contains only events recoverable from agent observations; unavailable event channels are omitted. The latent and event losses use the corresponding prediction heads. No hidden-state label or evaluator rollout enters this online score. A large belief update identifies potentially diagnostic evidence, but does not by itself justify changing the rule.

Proposal is triggered by persistent unexplained outcomes, a diagnostic event boundary, or failure to retrieve a structurally compatible explanation. Delayed effects are assessed at their predicted time: an unchanged indicator before the expected delay expires is not itself a failed transition. The proposal threshold is calibrated on development trajectories containing both ordinary state changes and observation disturbances.

C.4 Template-Guided Candidate Induction

A proposal uses a trailing window of 5 to 20 recorded interactions, depending on the mechanism family, with actual timestamps retained. At proposal time t_0, every fitting observation belongs to h_{t_0}; subsequent observations are reserved for validation. The template head scores the nine forms in Appendix B.3 using history, belief changes, recent actions, and retrieved entries. Variable and hook proposals combine type compatibility with diagnostic state changes and the controls involved in the segment.

Parameter fitting is constrained by the proposed template. Discrete candidates estimate transition mappings; threshold candidates estimate a threshold and persistence requirement; timer candidates estimate event lag; rate and accumulator candidates estimate quantitative dynamics; stochastic candidates estimate outcome probabilities. The candidate also specifies any changed binding or precondition. Latent-state trajectories can guide fitting, but they are inferred evidence, not evaluator state at deployment.

Once fitted, a candidate records its parent version, fitting cutoff, editable fields, and initial evidence state. Its parameters are frozen during the next validation interval. A further refit creates a new candidate version and a new validation interval; previously observed outcomes do not become fresh evidence for the refitted candidate.

C.5 Fresh-Evidence Validation and Scoped Writes

The unchanged rule is evaluated against each candidate using the same subsequent actions and observations. Both hypotheses maintain their own filtered state and predict each outcome before receiving it. All other program rules and neural weights are held fixed in this comparison. For a validation segment S, define

$$L_{\text{obs}}(\mathcal{P}; S) = \frac{1}{|S|} \sum_{s \in S} \left[\ell_x(x_{s+1}, \hat{x}^{\mathcal{P}}_{s+1|s}) + \lambda_y \ell_y(y_{s+1}, \hat{p}^{\mathcal{P}}_{s+1|s}) \right]. \tag{9}$$

The predictions depend on h_s, the executed action and elapsed time, and the fixed candidate version; they do not use o_{s+1} until scoring and the subsequent filter update. Missing observations are omitted from the corresponding loss, rather than treated as negative events. A candidate and its unchanged reference use the same observable channels and loss normalization.

The edit score follows the equation for validated repair provided in the subsection Sec. 3.3:

$$\Delta(e) = L_{\text{obs}}(\mathcal{P}_r; S) - L_{\text{obs}}(\mathcal{P}_r \oplus e; S) - \lambda_e \text{size}(e). \tag{10}$$

The size penalty accounts for changed fields and expanded scope. Acceptance requires valid types and bindings, $\Delta(e) > \delta_{\text{ins}}$, and at least n_{ins} supporting diagnostic segments. The margin and support requirement are development-set settings rather than test-session choices. Support and counterexamples refer to distinct diagnostic segments; repeatedly scoring overlapping copies does not increase support. For stochastic programs, contradictory evidence is assessed through predictive loss, rather than counting every unsuccessful action as a wrong rule. If fresh evidence does not favor a change over the unchanged rule, the candidate remains uncommitted.

Binding-scoped repair. An accepted edit creates a new instance version for its schema and device binding. Its parent is retained for provenance, while the shared schema and other device bindings remain unchanged. Instance-specific confidence and counterexamples are also kept local. A candidate whose parent has been superseded must be revalidated against the new reference. For template changes, shared state variables retain their meaning and new internal variables are inferred from the available history or their uncertain initialization; accepting an edit does not set a desired hidden outcome to true.

The controlled repair variants change this procedure at specific points: belief-only updating disables rule commits, parameter-only repair restricts the candidate class, no-validation repair bypasses the fresh-evidence gate, and shared-schema overwrite changes the commit destination. These interventions isolate evidence validation and write scope while retaining the surrounding inference and grounding modules. Their protocols and results appear in Appendix F.6.

C.6 Promotion and Consolidation

Accepted programs enter as provisional entries and incur a planning uncertainty penalty. Promotion requires at least n_{trust} supporting segments and a counterexample rate below $r_{\max}$, with thresholds selected on development data. Evidence updates apply to the version that generated the prediction, preserving the distinction between an old rule's failure and a new rule's support.

Consolidation operates outside the action-selection loop. MERGE combines compatible entries; SPLIT separates systematic context-dependent failures; ABSTRACT replaces instance identifiers with typed placeholders; EDIT refits local parameters; PRUNE removes unhelpful entries; and RE-INDEX refreshes retrieval keys and structured indices. Changes to shared schemas require evidence across their affected bindings, rather than treating one instance's repair as a global update.

Consolidation is scheduled every 500 training episodes or 200 provisional insertions, whichever occurs first. A ten-episode adaptation session does not itself trigger consolidation. Short-horizon repair and longer-term library maintenance are evaluated separately, and schema changes occur only when the configured consolidation trigger fires.

Algorithm 1 Online inference with executable programs and selective repair

Require: Goal g, session library $\mathcal{L}$, pending candidates $\mathcal{Q}$, frozen θ, prior b_0^-
Initialize episode history h_0 and forecast cache $\mathcal{F}_0$ using the reset marker
Reset candidate and reference state filters while retaining their rule evidence
for $t = 0, 1, \ldots$ until termination or budget exhaustion **do**
    $x_t \leftarrow \mathrm{Enc}_\theta(o_t)$
    $R_t \leftarrow \textsc{RetrieveBindRank}(\mathcal{L}, h_t, x_t, b_t^-, g)$
    $b_t \leftarrow q_\theta(m_t \mid h_t, R_t)$
    if $t > 0$ **then**
        Score cached forecasts $\mathcal{F}_t$ and compute ϵ_{t-1}
        $\mathcal{E}_t \leftarrow (h_t, R_t, b_t, \mathcal{F}_t)$
        $(\mathcal{L}, \mathcal{Q}) \leftarrow \textsc{UpdateLibrary}(\mathcal{L}, \mathcal{Q}, \mathcal{E}_t, \epsilon_{t-1})$
        if an active retrieved version changed **then**
            Refresh R_t and refilter b_t using the updated program context
        end if
    end if
    Propose K operations $\tau_t^{(k)} = (\mathrm{op}, \mathcal{C}, \beta, \eta)$
    for $k = 1, \ldots, K$ **do**
        $\hat{b}^{(k)} \leftarrow \mathrm{Exec}_\theta(b_t, R_t, \tau_t^{(k)})$
        Score $\tau_t^{(k)}$ using the $S_\theta(\tau)$ scoring equation from Sec. 3.4
    end for
    Select the highest-scoring $\tau_t^\star$ and decode an action chunk
    Select its execution prefix and cache state and observable predictions in $\mathcal{F}_{t+1}$
    Cache corresponding predictions for pending candidates and their unchanged references
    Execute the prefix, record actual commands and elapsed time, and receive o_{t+1}
    Extend h_{t+1}; retain the predictive belief as b_{t+1}^-
end for
Process the final observed transition; return $(\mathcal{L}, \mathcal{Q})$ for the next episode

C.7 Online Inference and Library Update

Algorithm 1 gives the receding-horizon loop for one episode, and Algorithm 2 specifies the update it calls. Within an adaptation session, the library and pending candidate evidence are passed between episodes. Predictions are conditioned on the executed action prefix and elapsed time, rather than on an un-executed suffix or the intended complete chain. If execution is truncated, predictions are aligned to the recorded commands using the saved pre-action belief, without conditioning on the new observation. Pending candidates are evaluated on new observations even when no new proposal is triggered. If an accepted edit changes a retrieved instance, the agent refreshes its retrieved context and belief before selecting the next operation.

C.8 Training and Evaluation Boundaries

Perception, belief inference, query encoding, binding and template scores, transition heads, operation scoring, and action grounding are trained with the objective in Appendix E.2. Retrieval selection, candidate admission, version commits, promotion, and consolidation are discrete operations. Training annotations may supervise program induction and validation; deployed repair uses only the agent's observations, action records, goals, and memory. Hypothetical evaluator queries are read-only and neither provide online validation labels nor mutate the control library.

At an episode reset, physical-state belief and temporal progress are reinitialized according to the agent-visible reset protocol, while permitted mechanism memory persists within an adaptation session. Reference and candidate filters receive the same reset marker; resets are not scored as ordinary mechanism outcomes. Independent sessions restore the trained library and clear candidate evidence and overrides learned during the preceding session. The fitting routines and validation/promotion settings are fixed from development data and recorded with the training configuration.

Algorithm 2 Fresh-evidence validation and binding-scoped library update

Require: Library $\mathcal{L}$, pending candidates $\mathcal{Q}$, evidence $\mathcal{E}_t = (h_t, R_t, b_t, \mathcal{F}_t)$, surprise ϵ_{t-1}

Update active-version evidence using previously cached predictions and the new observation
for each $e \in \mathcal{Q}$ with an unchanged parent version **do**
    Score its cached prediction and the reference prediction on the same new evidence
    Update its validation loss, support, and counterexamples
end for
Retire superseded candidates and collect candidates passing structural and evidence checks
if at least one candidate passes **then**
    Select the candidate with the highest admissible repair score
    Commit a provisional override at its schema and device binding
    Refresh affected retrieval entries and retire candidates with superseded parents
end if
if a diagnostic proposal trigger fires **then**
    Extract a trailing fitting segment ending at t
    Propose typed templates, scopes, hooks, bindings, and preconditions
    Fit local parameters and reject structurally invalid candidates
    Add frozen candidate versions to $\mathcal{Q}$ with empty future-validation records
end if
Promote eligible provisional entries using additional support and counterexample rates
if the consolidation schedule fires **then**
    Queue validated consolidation and index refresh outside the control step
end if
return $(\mathcal{L}, \mathcal{Q})$

D MechanismBench Specification

MechanismBench separates the hidden transition system from the observations supplied to the agent. This section specifies task generation, structural splits, intervention queries, and adaptation sessions used in Sections 3.7 and 4. The template semantics are given in Appendix B.3; here we define how they become evaluation tasks.

D.1 Task Specification and Information Boundaries

A task specifies entities and controls, action bounds, visible and hidden variables, local programs, an observation map, an initial-state distribution, a goal, and an episode budget. Split and query annotations are stored separately from agent inputs. Program declarations include units, reset conditions, internal clocks, and persistence state, so that a task's future is determined by its full state, actions, and stochastic draws.

The generator first constructs a typed program-dependency graph, then assigns parameters, scene bindings, and a visual domain. Edges connect effects to preconditions. Syntactic checks reject incompatible bindings and unresolved simultaneous writes; evaluator rollouts check that the goal is reachable within the budget. Local transition rules are shared across visual realizations, while changes to appearance are applied through the observation map.

Agent interface. The deployed agent receives observations, its executed actions and their durations, the goal, and episode-boundary markers. Available control names and action ranges describe the interaction interface, not the hidden mapping from controls to effects. Symbolic fields, when present, contain only observable quantities. Predicted attributes extracted from images are distinguished from evaluator state. Failed interventions consume their normal action duration and budget; the interface does not reveal a hidden failed precondition.

Evaluator interface. The evaluator additionally accesses the full mechanism state, active rule parameters, event queue, program graph, and random-number state. These quantities support training annotations, task verification, and scoring. At test time, they are excluded from policy inputs, retrieval

keys, generated-code environments, and online repair validation. Diagnostic queries operate on copies and cannot alter the control trajectory or its memory.

Goals and names. Goals specify outcomes, such as a requested light output or successful dispensing, without supplying the required mechanism sequence. The same predicate-to-token mapping is used across methods. An opaque-name control replaces predicate and entity names through a fixed, consistent bijection across training and evaluation; it changes naming rather than asking the agent to interpret an unseen goal vocabulary. Neither readable names nor task identifiers encode thresholds, hidden initial states, or edit types.

D.2 Structural Splits

Task assignments are made before extracting training windows or populating memory. Structural keys retain template identities and dependency links while canonicalizing entity identifiers, preventing a renamed copy of a held-out chain from entering the training library.

In-distribution and observation aliasing. In-distribution tasks share mechanism forms and depth ranges with training but vary scene instances, parameters, and initial states. The aliasing set pairs histories with similar visible observations and different action-relevant hidden states. The latter histories remain distinguishable through their preceding interactions; the test concerns retention of evidence rather than recovery of information never observed.

Composition and decomposition. Composition holds out complete dependency chains while retaining their local program forms in training. Depth counts the required mechanism transitions along a goal-enabling chain, including relevant autonomous events, rather than motor commands or elapsed ticks. Repeated waiting does not increase depth. Decomposition instead presents subchain goals independently after training exposure within larger tasks, with initial conditions that make the subproblem executable. The two protocols have separate task assignments: a chain held out for composition is not introduced through decomposition training. Composition scores give equal weight to depths two through six.

Nuisance variation. Textures, colors, object skins, layouts, labels, distractors, and indicator renderings vary while the transition graph and mechanism parameters remain fixed. Observable outcome semantics remain available through the task's observation interface. This separates changes in appearance and perceptual binding from changes to the transition law.

Local rule changes. Edits change one local transition object: a numerical parameter, control mapping, cycle order, precondition, or compatible template. Template substitutions preserve the relevant variable and effect types and are checked for reachability. Changed observation conventions are evaluated as nuisance variation rather than silently included as rule changes. Other mechanisms remain unchanged, although their downstream behavior can legitimately change when they depend on the edited mechanism.

D.3 Prediction Queries

Prediction queries specify both the intervention and what happens afterward. The alternate-action interface is

$$Q_t = (h_t, a'_t, \Delta_t, \pi_{\text{cont}}, H_q, Y), \tag{11}$$

where a'_t and Δ_t specify the candidate command and duration, π_{cont} is a supplied continuation rule or action sequence, H_q is the evaluation horizon, and Y is the outcome predicate. A no-op continuation explicitly allows active processes and timers to evolve. The model predicts the outcome distribution from its history and memory, without observing the cloned state.

Alternate actions. The evaluator clones the complete simulator state, including internal clocks and pending events, substitutes the candidate action, and executes the specified continuation. Histories and queries form a fixed bank shared by all methods; they are not selected from each evaluated policy's own visitation distribution. Sampling covers diagnostic stages such as blocked actions,

Table 4: **MechanismBench episode allocation.** Query clones are additional evaluator computations, not additional agent interaction episodes. Adaptation episodes are grouped into sessions for memory retention and statistical aggregation.

Partition	Episodes
Training	12,000
Validation	2,000
In-distribution test	2,000
Observation-aliasing test	1,500
Composition and decomposition	1,500
Nuisance-shift test	1,500
Adaptation: 150×10 episodes	1,500

pending delays, and near-threshold progress. Class frequencies and sampling weights are retained with the query bank.

Specified state or rule interventions. A hypothetical state assignment or program edit is supplied as part of its query, together with its binding and any required reset semantics. The evaluator applies that same specification to a clone. State assignments must respect type and clock invariants; a rule edit alone does not retroactively change accumulated progress or undo an already latched effect. These diagnostics are reported separately from alternate-action accuracy and from adaptation to an unannounced change.

Stochastic outcomes. For stochastic mechanisms, models predict probabilities and evaluator rollouts sample from the intervened dynamics. Accuracy is accompanied by probabilistic scoring rather than treating an unlikely individual outcome as a deterministic model error. Shared random draws can reduce variance in paired evaluator comparisons, but the reported prediction task concerns interventional outcome distributions. Same-world counterfactual coupling is not assumed merely because the initial state was cloned.

D.4 Adaptation Sessions and Resets

An adaptation session starts from the trained checkpoint and its initial memory, followed by an unannounced local rule change. The agent receives ten post-change episodes. Physical mechanism state resets between episodes while the changd rule persists. Permitted agent-side knowledge—program versions, supporting transitions, recurrent context, or notes—persists within the session. The reset marker lets each method distinguish a new physical initialization from evidence of another rule change.

Neural network weights remain fixed. CausalWAM updates program entries; programmatic baselines may revise code and fit their non-neural model parameters; recurrent and multimodal baselines retain their allowed context. Independent sessions restore the trained memory and clear test-session evidence. The agent receives no edit identifier, changed parameter, or evaluator-generated validation outcome.

The closed-loop adaptation curves use each method's own interactions. The selective-repair study instead replays shared exposure histories before read-only evaluation, isolating the update procedure from evidence gathering. Its state-only and observation-disturbance controls do not change transition rules. These protocols remain separate in both aggregation and interpretation.

D.5 Dataset Accounting

The benchmark uses 120 base program schemas drawn from the nine templates and four visual domains: appliance, game terminal, lab device, and robot fixture. Tab. 4 gives the episode allocation. Composition and decomposition form a combined generalization allocation with separate manifests; a listed episode count is not a count of independent counterfactual queries. Adaptation uses 150 sessions of ten post-change episodes. Interaction sequences range from 10 to 60 steps, with mean length of 28.3; elapsed time is recorded separately from step count.

Table 5: **Core execution settings.** Task depth, candidate-chain length, action horizon, and mechanism rollout horizon are distinct quantities.

Setting	Value
Action horizon H, simulation / Franka	$8 / 4$
Candidate operations K	8
Maximum candidate-chain length D	4
Mechanism rollout horizon H_m	5
Vector candidates K_R / retained programs K'_R	$16 / 5$
Evidence window, discrete / time-dependent templates	$8 / 20$
Consolidation interval, training episodes / insertions	$500 / 200$
Belief samples per candidate rollout	32
Probe hidden width	256
Independent evaluation runs	5

Each evaluation record identifies its task instance, structural split, depth, initialization, and episode or adaptation-session index. The same task manifests are used across methods. Train/validation separation applies to program-support segments and initialization libraries as well as neural training examples. Hypothetical queries, adaptation curves, and representation diagnostics retain separate identifiers so that their metrics are not inadvertently pooled.

E Training, Baselines, and Metrics

This section supplies the training objective condensed in Section 3.6, the comparison interfaces in Section 4, and the hidden-state diagnostics summarized in the main results. It distinguishes training supervision from information available during control.

E.1 Training Schedule and Model Interfaces

Training has three stages. Trajectory pretraining learns the perceptual bottleneck, recurrent inference, action-conditioned dynamics, action decoder, and future-latent head. Library bootstrapping induces local programs from training segments and validates them on disjoint training evidence. Joint fine-tuning then trains inference, proposal, binding, retrieval scores, operation selection, and grounding with the retrieved context. The discrete library lifecycle remains outside backpropagation.

The visual backbone supplies pretrained image features; the downstream bottleneck and recurrent state share their configuration across CausalWAM, WAM-direct, and Sup-WAM. Typed heads predict categorical state probabilities and Gaussian parameters for normalized continuous variables. Variable masks remove absent state components rather than treating them as negative labels. The model receives previous executed actions and elapsed time, not evaluator event logs.

To propagate uncertainty through discrete guards and continuous updates, the executor samples 32 states from the current typed belief and carries each sample through the complete candidate chain. Samples are shared across candidate comparisons, as are the indexed random draws for stochastic effects. Preconditions are tested on each intermediate sample, preserving dependencies created by earlier transitions. Empirical outcome frequencies and continuous moments provide the summaries used by the scoring heads. This is a numerical approximation to Equation 6, not access to simulator state.

We optimize with AdamW and a cosine learning-rate schedule, selecting checkpoints by validation task success, with alternate-action accuracy used to break ties. Training batches contain contiguous history windows; preceding observations initialize recurrence without contributing future information. Future targets are stop-gradient observation embeddings, and target padding is masked. Tab. 5 records the inference settings shared with the main method.

E.2 Training Objective

The joint objective is

$$\begin{aligned}\mathcal{J} = \lambda_{\text{belief}}\mathcal{J}_{\text{belief}} + \lambda_{\text{dyn}}\mathcal{J}_{\text{dyn}} + \lambda_{\text{prog}}\mathcal{J}_{\text{prog}} + \lambda_{\text{target}}\mathcal{J}_{\text{target}} \\ + \lambda_{\text{act}}\mathcal{J}_{\text{act}} + \lambda_{\text{future}}\mathcal{J}_{\text{future}} + \lambda_{\text{ret}}\mathcal{J}_{\text{ret}}.\end{aligned} \tag{12}$$

Each term is averaged over its valid variables, tokens, or prediction targets before weighting. We use unit weights for belief, dynamics, target selection, action, and future prediction, and weights of 0.25 for program and retrieval supervision. The same normalization and auxiliary targets are used for the supervision-matched comparison.

Belief and dynamics. Let $\ell_{\text{state}}(m, q)$ denote the sum of mean categorical cross-entropy over discrete variables and mean Gaussian negative log-likelihood over normalized continuous variables. Continuous normalization uses the declared variable domain rather than test-set statistics, with a minimum standard deviation fo 0.01 in normalized units to prevent degenerate likelihoods. We train

$$\begin{aligned}\mathcal{J}_{\text{belief}} &= \mathbb{E}\,\ell_{\text{state}}(m_t, b_t), \\ \mathcal{J}_{\text{dyn}} &= \mathbb{E}\left[\ell_{\text{state}}(m_{t+1}, \hat{b}_{t+1|t}) + \ell_{\text{obs}}(o_{t+1}, \hat{o}_{t+1|t})\right].\end{aligned} \tag{13}$$

Here ℓ_{obs} scores observation embeddings and agent-visible events, not generated pixels. The predictive belief uses the inferred current belief and recorded action, including its duration. Ground-truth state supplies a training target rather than replacing inference at the model input. Training alternate-action examples add the corresponding transition targets to this objective; their outcomes never appear in the history used to make the prediction.

Program induction. Program supervision combines categorical losses for template choice, variable binding, hook binding, and precondition structure, normalized parameter regression, and local transition likelihood. Program identifiers are matched through their typed structure and bindings rather than treated as arbitrary semantic labels. Labels are used to train proposal and fitting modules; online candidates are inferred and validated from observed transitions as in Appendix C.

Operation selection. Successful trajectories supply reference operations and executable chain prefixes. The loss trains operation and chain tokens, bindings, and continuous arguments, together with a softmax ranking loss on candidate scores S_θ. Equivalent program versions are matched by their transition signatures. Continuous arguments describe intended interventions, such as hold duration, rather than assignments to hidden state. Demonstration-derived targets are training supervision; at deployment the planner proposes and scores its own operations.

Action grounding. For a normalized demonstration chunk A and diffusion index ℓ, define $A_\ell = \sqrt{\bar{\alpha}_\ell}A + \sqrt{1-\bar{\alpha}_\ell}\varepsilon$, with $\varepsilon \sim \mathcal{N}(0, I)$. The conditional denoising loss is

$$\mathcal{J}_{\text{act}} = \mathbb{E}\left[\|\varepsilon - \varepsilon_\theta(A_\ell, \ell; b_t, x_t, R_t, g, \tau_t)\|^2\right], \tag{14}$$

using the masked mean over action dimensions and valid timesteps (Chi et al., 2025). In simulation, categorical command fields are encoded numerically and projected to legal commands at execution; continuous arguments are bounded by the shared action interface. Physical decoding produces the corresponding robot-control chunk. Every method's command adapter performs only interface validation and grounding, not hidden-precondition testing.

Future prediction. The future head predicts observation embeddings and typed mechanism state along the recorded action sequence. Its loss averages embedding regression and state likelihood over offsets two through H; the one-step terms are already included in $\mathcal{J}_{\text{dyn}}$. Targets are masked at episode termination, and rollout predictions do not receive future observations. This term supplies multistep action-consequence supervision; observation-latent predictions also provide online repair evidence without generating pixels.

Table 6: **Auxiliary supervision and adaptation memory.** State and program columns refer to labels used to train the control model, not evaluator-only probes. Neural weights remain fixed during adaptation. The state-assisted reference follows CausalWAM training but additionally receives current hidden state.

Method	State labels	Program/chain labels	Persistent session knowledge
Diffusion Policy	No	No	Observation–action context
Latent WM+MPC	No	No	Recurrent state and history
WAM-direct	No	No	Recurrent state and history
Sup-WAM	Yes	Yes	Recurrent state and history
PoE-MB	No	No	Expert programs, coefficients, and evidence
MM-agent	No	No	Interaction history and written notes
CausalWAM	Yes	Yes	Bound programs, state belief, and evidence

Retrieval. Contrastive query–program learning is combined with pairwise ranking of bound candidates. Positive entries explain the local transition or participate in a goal-enabling chain; negatives include surface-similar programs with incompatible hooks, effects, or parameters. A program is not labeled negative merely because an earlier operation must establish its preconditions. Retrieval supervision therefore matches the planner's intermediate-state applicability checks.

E.3 Parameter Fitting and Evidence Calibration

The template bounds the fitting problem. Discrete transition maps use posterior-weighted transition counts with unit pseudocounts. Threshold fitting searches ordered candidate boundaries between inferred state values and admissible persistence counts; delay fitting maximizes event-timing likelihood, treating unresolved pending effects as right-censored observations. Rate and accumulator coefficients use bounded least squares on inferred increments with a penalty toward the parent parameters. Stochastic gates use a Beta$(1, 1)$ prior with posterior-weighted success and failure counts. Neural transition heads remain fixed during online fitting.

Candidates are evaluated by the observable predictive loss in Equation 9, not by agreement with their own inferred hidden trajectories. Latent errors are normalized per feature and event errors per observed channel. The edit-size penalty counts changed fields after normalizing parameter changes by their declared ranges, with weight 0.01.

We calibrate proposal surprise at the 95th percentile of unchanged-rule development transitions. The insertion margin is calibrated on unchanged-rule development sessions by replaying proposal and validation with commits disabled and taking the 95th percentile of the largest candidate improvement per session. This accounts for repeated candidate testing in the calibration data without treating each attempted edit as an independent test. Admission additionally requires support from three non-overlapping diagnostic segments; promotion requires eight supporting segments and a counterexample rate below 0.2. Counterexamples are diagnostic segments exceeding the calibrated predictive-loss tolerance, not individual stochastic failures. These settings are fixed before the test sessions.

When retrieval fails to provide a parent program, the unchanged recurrent predictor is the reference for that local scope. Such proposals must improve observed prediction over leaving the mechanism implicit; missing retrieval is not sufficient for admission. Accepted new entries receive a binding-local version, and hypotheses depending on a superseded parent are revalidated.

E.4 Baseline Implementations and Information Access

Tab. 6 distinguishes per-trajectory training annotations from persistent test-session knowledge. All methods use the same observation and action contract and have access to the training trajectory collection. The Sup-WAM comparison additionally matches mechanism and program supervision. Diagnostic probes receive hidden labels only after their control models have been frozen.

Neural alternatives. Diffusion Policy uses action imitation without the dynamics or program losses (Chi et al., 2025). Latent WM+MPC learns recurrent action-conditioned dynamics and searches candidate action sequences. WAM-direct shares action and future-latent learning but decodes without executable program operations. Sup-WAM adds the same state, template, binding, parameter, and

operation targets as auxiliary heads on its recurrent representation; these predictions do not invoke an executor or an editable library. Thus supervision matching concerns the information supplied during training, while the tested difference is its computational use.

Programmatic world model. PoE-MB adapts the expert synthesis, product-of-experts prediction, coefficient optimization, and iterative refinement of PoE-World (Piriyakulkij et al., 2026). Its observation serialization contains visible or perception-derived attributes and missingness indicators, never simulator mechanism state. Control descriptions specify command syntax without revealing transition rules. Expert fitting and repair use training trajectories and allowed session evidence. Generated code runs without access to evaluator objects, clone APIs, or task-source files. Its planner remains a separate implementation rather than being replaced by CausalWAM's typed-chain planner.

Multimodal agent. MM-agent receives the goal, current observations, timestamped action history, and persistent notes. Its action interface exposes the same controls and argument bounds as the other methods. Notes may record hypothesized mechanisms but contain no evaluator feedback beyond the ordinary observations. The checkpoint, prompt, context-management rule, decoding settings, and retry budget are fixed across test conditions. Prediction queries use a read-only copy of the agent context and request an outcome distribution; their labels never enter the control conversation.

Component ablations. The no-planner variant retains belief and retrieved programs but directly conditions the action decoder on them. The no-library variant predicts transient local program hypotheses from its recurrent state; these are not committed as external entries. Raw episodic retrieval stores observation–action segments and infers temporary hypotheses from their context, rather than storing editable rules. Removing future co-training suppresses the multistep future loss while retaining the one-step observable dynamics needed for repair scoring. These variants retain the same control data and action interface.

Resource accounting. Programmatic and multimodal baselines can inspects training demonstrations through their frozen context-selection procedures, but test-task outcomes cannot be used for prompt or model selection. Environment interaction budgets are shared; internal search, code-generation calls, and token use are recorded separately rather than asserted to be identical across architectures. Exact pretrained checkpoint identifiers, prompts, and resource limits are associated with each run configuration. Fixed language-model evaluations use repeated runs, not claims of independently retraining the language model.

E.5 Diagnostic Protocol and Hidden-State Results

Hidden-state diagnostics use the common observation-aliasing history bank. Recurrent and latent representations are computed from the available history without future inputs. For models without typed state outputs, we train a frozen-representation probe with one hidden layer of width 256, ReLU activation, and separate categorical and continuous heads. Probe training uses only the training partition and evaluator labels; it never updates the controller or supplies features to planning. Typed models are evaluated through their native belief heads. Both routes are reported explicitly because supervised state recovery and state recoverable by a post-hoc probe are different interfaces.

Tab. 7 contains the state diagnostics summarized in the main text. Improvements in discrete F1, continuous error, and calibration show that the reported prediction gains accompany better hidden-state estimates rather than only a different action-selection rule.

Alternate-action prediction is evaluated only for methods with an explicit prediction interface. Neural world models use their action-conditioned outcome heads, PoE-MB uses its predictive expert distribution, and MM-agent returns query-conditioned probabilities. Outcome-head training uses the same training-query collection where applicable; a pure action-only policy is not assigned a native counterfactual score through an unspecified auxiliary model.

E.6 Metrics and Aggregation

Success and generalization. Task success is binary goal completion within the specified budget. Composition and decomposition use their own task assignments; composition averages success

Table 7: **Hidden-state diagnostics on aliased histories.** Implicit-state baselines use frozen probes; CausalWAM variants use their typed belief outputs. Values are means and SEM over five runs. The state-assisted reference is omitted because its current state is supplied directly.

Method	Hidden-state F1 ↑	NRMSE ↓	ECE ↓
Diffusion Policy + probe	58.7 ± 1.4	0.241 ± 0.012	0.128 ± 0.009
Latent WM+MPC + probe	68.9 ± 1.1	0.188 ± 0.010	0.091 ± 0.007
WAM-direct + probe	73.6 ± 0.9	0.163 ± 0.008	0.076 ± 0.006
No program library	85.4 ± 0.6	0.092 ± 0.005	0.045 ± 0.004
CausalWAM	$\mathbf{88.8 \pm 0.5}$	$\mathbf{0.076 \pm 0.004}$	$\mathbf{0.036 \pm 0.003}$

equally across depths two through six. Nuisance-shift success uses the unchanged-program evaluation manifest. Metrics are first computed within each run and then summarized across runs.

State recovery and calibration. For each discrete variable, macro-F1 is computed over its state values after pooling its valid evaluation records, then averaged across variables. It is not an average of per-frame F1 scores. Continuous error is the mean variable-wise RMSE divided by that variable's declared range. ECE uses 15 equal-width confidence bins, comparing top-class confidence with accuracy within each variable before averaging variables. Missing variables and padded steps are masked. F1 is reported as a percentage; NRMSE and ECE are on their native scales.

Intervention prediction. Accuracy is the fraction of outcome queries whose highest-probability class matches the evaluator outcome. Alternate-action, specified state-intervention, and specified rule-edit queries are reported separately. For stochastic outcomes we additionally compute negative log-likelihood and Brier score on the same query distribution. Outcome labels from diagnostic calls remain evaluator-only, including after adaptation.

Valid operations and retrieval. An operation is valid if its bound intervention belongs to the evaluator's admissible goal-directed set, allowing equivalent chains and arguments. Valid-operation rate is

$$\mathrm{ValidOpRate} = \frac{100}{N} \sum_{t} \mathbf{1}[\tau_t \in \mathcal{T}_t^{\mathrm{valid}}]. \tag{15}$$

This is a set-membership rate, not macro-F1 over program identifiers. Retrieval P@5 counts relevant bound entries among the final five positions, with unfilled positions counted as misses. Relevance is determined by transition compatibility and the current or goal-enabling mechanism context, not by visual similarity. Duplicate versions of the same resolved entry do not earn repeated credit. The number retrieved per decision is distinct from unique entries used across an episode.

Adaptation and repair. For run r, let $s_{r,i}$ be percentage success at post-change episode i, averaged over its sessions. Then

$$\mathrm{AUC}_r = \frac{1}{10} \sum_{i=1}^{10} s_{r,i}. \tag{16}$$

AUC uncertainty is computed from run-level AUCs, not by averaging the ten pointwise error bars. Controlled-repair success is evaluated after shared exposure and is not pooled with closed-loop episode success. False-commit rate is the percentage of unchanged-rule control sessions containing an accepted rule edit. Unaffected-device retention is the paired difference in its success before and after repair, measured in percentage points.

Uncertainty and paired comparisons. For run-level measurements $v_1, \ldots, v_n$, we report $\bar{v}$ and $\mathrm{SEM} = s(v)/\sqrt{n}$, with $n = 5$ for the simulation comparisons. Shared task manifests permit paired method differences. Episodes within an adaptation session and queries from one cloned history are not treated as independent training runs. Physical counts and evaluation blocks are reported separately in Appendix G.

Table 8: **Generalization through local program reuse.** Success percentages are reported on distinct composition, decomposition, and nuisance-shift splits. CausalWAM's gains extend beyond composing longer chains to reusing their parts and operating under changed appearance.

Method	Composition ↑	Decomposition ↑	Nuisance shift ↑
Diffusion Policy	37.8 ± 2.3	48.6 ± 2.1	56.7 ± 2.0
Latent WM+MPC	42.5 ± 2.1	52.9 ± 2.0	60.8 ± 1.8
WAM-direct	46.3 ± 1.9	55.6 ± 1.8	64.1 ± 1.7
Sup-WAM	61.1 ± 1.0	68.2 ± 1.6	74.1 ± 1.3
PoE-MB	64.9 ± 1.1	71.3 ± 1.6	76.2 ± 1.0
MM-agent	53.8 ± 1.3	63.2 ± 1.9	69.4 ± 1.3
No program library	59.7 ± 1.6	68.4 ± 1.4	76.6 ± 1.3
Raw episodic retrieval	63.4 ± 1.5	70.8 ± 1.3	73.5 ± 1.4
CausalWAM	$\mathbf{76.2 \pm 1.2}$	$\mathbf{82.1 \pm 1.0}$	$\mathbf{84.8 \pm 1.1}$
State-assisted reference	83.5 ± 0.9	88.6 ± 0.8	89.2 ± 0.8

E.7 State-Assisted Reference and Leakage Controls

The state-assisted reference receives current m_t, but not true future outcomes or the identity of an unannounced rule change. Its control and prediction scores provide a reference for the effect of state uncertainty; they are not a guaranteed optimum. Current-state inference metrics are omitted because supplying their target makes that comparison uninformative.

Training annotations, probe labels, cloned outcomes, and reset instrumentation are restricted to their declared roles. Agent-side state estimates and code environments cannot access them during control. Independent test sessions restore model and memory state; hypothetical queries operate on copies. Every aggregate is associated with its task manifest, method configuration, and unit of repetition, preventing a protocol change or repeated quantity from silently receiving a different denominator or uncertainty definition.

F Additional Results and Design Diagnostics

This section expands the capability comparisons and controlled studies in Section 4. Generalization and closed-loop adaptation measure the complete system; matched-budget planning, executable-rule interventions, and shared-evidence repair isolate particular uses of the program interface. Unless a table specifies otherwise, simulation results are means and SEM over five runs, using the aggregation rules in Appendix E.6.

F.1 How Well Do Programs Transfer across Tasks and Appearances?

Tab. 8 separates unseen complete chains, independently reused subchains, and nuisance appearance changes. The task assignments follow Appendix D.2; none of these evaluations changes the test-time information contract. Composition is averaged equally across depths, whereas decomposition and nuisance success use their respective task manifests. The state-assisted reference is separated because it receives privileged current state.

Episodic retrieval improves composition over removing memory, but its nuisance-shift success is lower than the no-library variant. The structured library improves both, supporting reuse of transition knowledge rather than retrieval of complete, appearance-specific interaction sequences. These results concern recombination within the template vocabulary; parameterization, binding, and visual variation determine the distinct task instances.

F.2 How Does Recovery Develop across Post-change Episodes?

Tab. 9 supplements Figure 4 with selected episodes and the ten-episode AUC. Each method collects its own evidence while retaining the memory allowed by Appendix D.4. An episode reset clears physical state but not the accumulated knowledge about the changed rule. Consequently, these curves measure both evidence gathering and use of that evidence. AUC is computed from all ten episode scores within each run, not by interpolating the displayed points or averaging their error bars.

Table 9: **Closed-loop recovery after unannounced rule changes.** Entries are success percentages at selected post-change episodes and mean success across all ten episodes. Program memory improves recovery over time rather than relying on advance knowledge of the edit.

Method	Episode 1	Episode 5	Episode 10	AUC ↑
WAM-direct	32.8 ± 2.2	43.5 ± 2.4	52.7 ± 2.5	43.9 ± 1.9
Latent WM+MPC	34.0 ± 2.2	45.8 ± 2.3	55.1 ± 2.4	46.5 ± 1.8
Sup-WAM	36.5 ± 1.9	58.3 ± 2.4	69.7 ± 1.3	57.9 ± 0.5
PoE-MB	39.7 ± 2.6	65.3 ± 1.5	72.0 ± 1.5	61.1 ± 1.9
MM-agent	35.9 ± 1.3	54.0 ± 1.6	63.5 ± 1.5	52.7 ± 0.5
No program library	35.1 ± 2.1	47.9 ± 2.2	58.4 ± 2.3	48.7 ± 1.8
Raw episodic retrieval	38.3 ± 1.9	59.7 ± 2.0	70.2 ± 1.8	59.1 ± 1.5
CausalWAM	$\mathbf{42.2 \pm 1.7}$	$\mathbf{75.8 \pm 1.4}$	$\mathbf{85.4 \pm 1.1}$	$\mathbf{73.0 \pm 1.1}$
State-assisted reference	68.0 ± 1.3	89.5 ± 0.8	93.3 ± 0.6	86.5 ± 0.7

Table 10: **Retrieval quality and downstream use.** Hybrid retrieval improves program relevance together with composition and adaptation. P@5 evaluates bound program entries; raw-episode retrieval is compared through task performance in Tab.s 8 and 9.

Retrieval	Program P@5 ↑	Composition ↑	Adaptation AUC ↑
Vector-only	66.8 ± 1.4	67.4 ± 1.2	62.9 ± 1.4
Structured-only	70.1 ± 1.2	69.0 ± 1.1	65.5 ± 1.2
Hybrid	$\mathbf{81.5 \pm 1.0}$	$\mathbf{76.2 \pm 1.2}$	$\mathbf{73.0 \pm 1.1}$

The gap between CausalWAM and the no-library variant widens as diagnostic experience becomes available. Episodic memory recovers part of this benefit, while explicit rule updates provide stronger recovery. The controlled repair study below separates the update mechanism from differences in the observations each policy gathers.

F.3 Does Structured Retrieval Improve Program Selection?

Vector-only retrieval ranks program candidates by their learned query–key similarity, retaining the binding checks needed to execute legal commands. Structured-only retrieval uses typed hooks, effects, preconditions, and binding compatibility without learned vector recall. Hybrid retrieval combines vector recall with structured reranking. All variants return at most five resolved entries and use the same downstream program interface. State-precondition likelihood is a ranking feature rather than a hard filter on programs needed later in a chain. Results are report in Tabl. 10

The joint improvement in relevance and task performance supports combining contextual recall with an explicit check of what a program can do. Retrieval precision alone would not establish this benefit: entries must also support useful predictions, chains, and updates.

F.4 Does Typed Composition Use the Search Budget More Effectively?

This experiment changes search organization while fixing the checkpoint, inferred state, retrieved rules, action primitives, grounding interface, and scoring function. Typed search proposes extensions through effect–precondition links. Flat search proposes sequences over the same bound primitives and parameter ranges without using those links to organize expansion. Both execute the same transition rules, including failed preconditions, active processes, and delayed events. Search settings are selected on validation tasks, and both methods use the same maximum candidate depth and mechanism horizon. We report result in Tab. 11

One budget unit advances one proposed primitive transition over the fixed-size belief-sample set used by the executor. A composed chain consumes one unit for each transition evaluated; it is not counted as one action merely because it was proposed as a chain. Shared prefixes count once when their cached predictions are reused, and rejected candidates consume any rollout computation already performed. The number of belief samples is identical between methods. Thus the comparison matches transition-model work; proposal and ranking overhead are not asserted to have identical wall-clock cost.

Table 11: **Matched-budget planning.** Typed search achieves higher success using the same transition-evaluation budget. These budgeted configurations differ from the default proposal-count setting in the headline comparison.

Search interface	Budget	Composition ↑	Depth-six success ↑
Flat primitive search	64	60.9 ± 1.4	42.8 ± 2.0
Typed chain search	64	73.2 ± 2.4	55.2 ± 3.5
Flat primitive search	256	68.7 ± 1.1	52.4 ± 1.5
Typed chain search	256	78.5 ± 0.8	67.2 ± 1.5

Table 12: **Executable-rule intervention.** Updating execution improves predictions and operation choices on affected queries while preserving agreement on unaffected predictions. All entries are percentages.

Internal update	Edited-outcome accuracy ↑	Valid-operation rate ↑	Unaffected agreement ↑
Tokens only; executor unchanged	63.1 ± 1.4	60.2 ± 0.3	99.1 ± 0.1
Tokens and executable rule	85.7 ± 0.6	81.1 ± 0.4	98.9 ± 0.2

The benefit persists when both planners receive a larger budget and is visible on the longest held-out chains. This supports effect–precondition matching as a useful allocation of search effort, rather than explaining the gain solely through additional transition evaluations.

F.5 Do Predictions and Decisions Use Executable Rule Semantics?

Program surgery changes an internal transition rule while holding the observation history, initial belief, goal, neural weights, and unrelated programs fixed. The evaluator supplies the identity and specification of a typed edit and applies it to a cloned task. The model receives that edit specification, but not the resulting outcome or a corrected hidden state. This is a specified hypothetical intervention, not the unannounced-change adaptation task. The test uses matched, bound entries so that it isolates execution rather than template discovery or retrieval coverage.

We compare two edits to the same initial model copy in Tabb. 12. The first updates the descriptive program tokens but leaves the executor's rule unchanged. The second updates both tokens and the executable rule. All other lookup and scoring inputs are fixed; evaluation is read-only, with matched stochastic draws where applicable.

Outcome-sensitive queries are selected by whether the evaluator's consequence changes under the edit, independently of either model's predictions. The operation-sensitive subset requires disjoint valid-operation sets before and after the edit, so retaining any previously valid operation cannot satisfy the new requirement. Unaffected queries involve mechanisms outside the edited dependency path. Agreement on these queries measures stability of predictions, separately from their correctness.

The improvement on affected cases, together with high agreement elsewhere, supports semantic use of the program rather than indiscriminate sensitivity to a changed token sequence. The test establishes a role for the edited rule in this interface; discovering the appropriate edit is evaluated separately.

F.6 Can the Model Repair One Mechanism without Disrupting Another?

The repair study uses devices A and B with different bindings to a shared program schema and independent physical dynamics. Four conditions distinguish state and rule changes: a state-only change on A, an observation disturbance without a rule change, a parameter change on A, and a compatible template replacement on A. Device B retains its original rule in every condition. Template changes preserve the relevant variable and effect types while changing the transition form.

Shared evidence and read-only evaluation. Each variant receives the same ten exposure episodes within a condition, including actions, durations, observations, and reset markers. Exposure covers the relevant hooks and delayed responses without revealing the changed rule. The updater must use the state-inference and fresh-validation procedures in Appendix C; evaluator state is not part of the

Table 13: **Selective repair under shared evidence.** Success follows ten shared exposure episodes and subsequent read-only evaluation. False commits count unchanged-rule control sessions with an accepted rule edit. Retention is the paired change in unaffected-device success, in percentage points; values near zero indicate little disruption.

	Post-exposure success (%)				
Update variant	State only	Parameter change	Template change	False commits (%)	Unaffected Δ success
Belief update only	90.2 ± 1.0	63.0 ± 2.6	52.3 ± 2.0	0.0 ± 0.0	0.00 ± 0.00
Confidence update only	89.3 ± 0.9	72.4 ± 0.7	61.3 ± 0.8	0.0 ± 0.0	-0.32 ± 0.04
Validated parameter edits	89.6 ± 0.9	84.4 ± 0.6	58.0 ± 2.3	2.8 ± 0.7	-0.70 ± 0.18
No edit validation	86.3 ± 1.0	81.5 ± 0.9	69.4 ± 1.7	15.4 ± 1.2	-3.30 ± 0.28
Shared-schema overwrite	90.5 ± 1.0	91.0 ± 0.2	82.0 ± 1.3	3.2 ± 0.2	-12.68 ± 0.63
Binding-scoped repair	90.5 ± 1.0	91.0 ± 0.2	82.0 ± 1.3	3.2 ± 0.2	-0.30 ± 0.13

Table 14: **Planner sensitivity on in-distribution deep tasks.** Entries are mean success percentages. Performance improves with additional proposals, composition depth, and prediction horizon, with smaller changes beyond the defaults $K = 8$, $D = 4$, and $H_m = 5$.

K	Success	D	Success	H_m	Success
1	75.6	1	68.4	1	78.9
4	82.1	2	78.1	3	83.2
8	84.5	4	84.5	5	84.5
16	85.0	6	84.9	8	84.6

replay. After exposure, we freeze memory writes and evaluate goal-directed control under matched probe initializations. Probe outcomes neither validate pending candidates nor create further edits. Changed-device success therefore measures what the fixed exposure taught the updater, rather than how effectively its own policy collected evidence.

Update controls. Belief-only updating freezes program parameters and confidence. Confidence-only updating can change evidence metadata but not a transition rule. Validated parameter edits retain the template, bindings, and precondition structure while fitting its numerical parameters. The no-validation variant retains type and binding checks but commits fitted candidates without the fresh-evidence gate. Shared-schema overwrite and binding-scoped repair use the same fitted and validated edits, differing only in their destination: the shared schema or the instance override. Consolidation is disabled throughout this controlled assay, so it cannot account for differences in short-horizon repair.

In the Tab. 13, state-only success remains similar for the validated variants, while parameter-only repair is insufficient for a changed transition form. Removing validation increases false commits; changing the commit destination preserves changed-device recovery but affects retention on B. The write-scope variants apply the same accepted repair to A. Retention evaluates the full retrieval and control pipeline, including competition among library candidates; it is a behavioral metric, not an equality check of B's stored rule. These controls distinguish repair expressiveness, evidence validation, and binding scope.

F.7 How Much Search Is Needed?

Tab. 14 varies one planning setting at a time while holding the remaining settings at their defaults. These are deep tasks from the in-distribution control evaluation, not the held-out depth-six composition tasks in Figure 4. The two scores therefore have different denominators and should not be compared as measurements of the same split.

Vector recall shows a similar tradeoff: increasing K_R from 8 to 16 raises composition success from 74.1% to 76.2%, whereas $K_R = 32$ gives 76.5%. The final retained set remains capped at five in this comparison. Short planning horizons can solve longer tasks through repeated observation and replanning; they need not represent the entire task in one chain. The sensitivity results support the default allocation without establishing a wall-clock advantage over other model architectures.

Table 15: **Instrumented physical mechanisms.** Each fixture tests a distinct dependency between interaction history and an effective next action. Controller logs are evaluator-only; policy inputs include camera observations, proprioception, executed actions, timing, and the goal.

Fixture	Hidden mechanism	Visible evidence	Interaction and goal
Hidden-mode panel	Mode index, control mapping, and indicator delay.	Panel appearance and aliased or delayed indicators.	Press or hold controls to reach the requested output state.
Accumulator–threshold	Continuous progress, effective rate, threshold, and readiness state.	Control pose and sparse progress or readiness indications.	Set input, allow accumulation, then invoke the enabled affordance.
Delayed trigger	Pending-event state, remaining delay, and enabled affordance.	Control appearance and the delayed indication when it occurs.	Arm the trigger, wait, and perform the subsequent interaction.

G Physical Evaluation Details

The physical study uses a Franka arm and the three instrumented fixtures in Figure 6 to test correctly ordered and timed actions. Physical controls provide observable consequences, while controller logs define hidden state for evaluation and reproducible resets.

G.1 Fixture Construction and Digital Fabrication

We construct custom instrumented appliance fixtures from 3D-printed and laser-cut components assembled with physical controls, indicators, and fixture electronics, with pad-printed text labels. The enclosures provide task context and repeatable robot contact, while the controllers implement hidden modes, progress variables, and delayed effects. This construction supports economical iteration, controlled rule changes, and reproducible resets without replacing an entire appliance. For physical reproducibility, we will share 3D assets and controller specifications as part of the data. Our fixtures evaluate physical interaction with the specified control logic: the robot manipulates buttons and knobs, and hidden controller state is logged separately. An accumulator can therefore represent charge or temperature progress through controller dynamics while still requiring the robot to establish the correct physical inputs and timing.

G.2 What Each Fixture Measures

Tab. 15 separates hidden state, visible evidence, and goal-relevant interventions. Delayed or aliased indicators supply evidence through the observation map; their underlying modes and timers are not additional policy inputs.

The hidden-mode controller logs mode and button events. The accumulator fixture logs progress through its controller or encoder interface. The delayed-trigger fixture records invocation, scheduled effect, and activation times. Continuous evolution uses physical elapsed time, including motion, waiting, and policy computation, so pending processes continue while the agent deliberates.

G.3 Agent Interface and Instrumentation Separation

The agent receives ordinary camera and proprioceptive observations, action records with timing, and the goal. Fixture programs, hidden progress, scheduled effect times, and private button-event logs are withheld. An indicator visible in an image is available to the perception model; the corresponding controller value is not injected into its input. Observation and command timestamps align evaluator logs after execution without adding future observations to the agent's history.

The learned action grounder uses horizon $H = 4$ and executes a prefix before incorporating the next observation. Every compared method uses a fixed, declared robot-control interface and the same permitted controls, command bounds, and safety constraints. A command adapter may execute motion or enforce physical limits, but it does not test hidden task preconditions, wait for an unobserved readiness flag, or complete a missing mechanism sequence. Controller-detected input events drive

the fixture, so sending a press command without making contact does not count as a successful intervention.

G.4 Trials, Resets, and Success Accounting

The revised main-table protocol uses three evaluation blocks of 20 trials per method and fixture, giving 60 trials per fixture and 180 across the three fixtures for each method. Learned-policy blocks use separate checkpoints; a fixed pretrained multimodal agent retains its checkpoint across blocks, with independent trial contexts. Block and checkpoint identifiers remain distinct, since repeated deployment of one pretrained model is not retraining that model.

Initial mode offsets, progress, thresholds, and delays are sampled from the same evaluation distribution across methods. Trial order is balanced across methods to reduce session effects. Each fixed-rule success trial restores the starting agent memory and the fixture state. A scripted reset clears clocks, pending events, and progress, restores movable components, and returns the robot to its home configuration with the prescribed initialization perturbation. Reset information supplied to the policy is limited to the episode-boundary marker and ordinary observations.

A trial ends on goal completion, a safety termination, or the budget of 60 seconds or 25 high-level action steps, whichever occurs first. Waiting and sustained actions consume elapsed time, and rejected or ineffective commands consume their allotted budget. Policy-caused safety terminations count as failures. A fixture or recording malfunction is identified independently of the method's task outcome and logged separately from a policy failure. No human supplies a corrective task action during a counted trial.

For fixture f, let k_{fb} successes occur among n_{fb} trials in block b. The pooled success rate is

$$\hat{p}_f = 100 \frac{\sum_b k_{fb}}{\sum_b n_{fb}}. \tag{17}$$

With equal block sizes, this equals the mean block percentage. The main table reports integer success counts and SEM across the three block percentages, using the definition in Appendix E.6. Per-fixture percentages are not rounded first and then used to reconstruct counts. Any aggregate across fixtures gives equal weight to the three fixture types.

G.5 Instrumented Diagnostics and Rule-change Sessions

Goal success is the primary physical metric; hidden-state diagnostics use synchronized controller logs after the agent's predictions have been recorded. Discrete modes, latches, and event states use the variable-wise F1 definition in Appendix E.6. Continuous progress uses its declared range for NRMSE rather than an arbitrary discretization chosen after evaluation. These diagnostics are kept separate from the task-success denominator.

Rule-change sessions follow a different reset contract from independent success trials. A local cycle order, threshold, control mapping, or delay changes, and that rule persists through ten post-change trials while physical state is reset between trials. Agent memory persists within the session, neural weights remain fixed, and no edit specification is supplied. Independent sessions restore the initial checkpoint and memory. Adaptation AUC is computed per session and run, rather than mixing post-change trials with the fixed-rule success evaluation.

G.6 Safety and Interpretation of the Physical Evidence

All physical trials use human supervision and an emergency stop, with workspace, velocity, force/torque, and gripper constraints enforced independently of the learned model. Invalid commands are clipped or rejected by the low-level controller. These safeguards limit physical execution; they do not reveal which hidden mechanism operation would achieve the goal.

The fixture images and the appliance trace in Figure 5 illustrate how belief, retrieval, and actions follow successive mechanism stages. The success counts and controlled studies supply the quantitative evidence. Successful interactions add program evidence; they do not individually establish promotion or a validated rule change.