PREFIXGUARD: ONLINE FAILURE WARNING AND TRACE-GROUNDED DIAGNOSIS FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents execute multi-step tasks, but terminal verdicts arrive too late for intervention. Agent traces mix messages, tool calls, and feedback, making it difficult to learn failure signals from terminal outcomes. Even if a model learns to predict failure from execution prefixes, its predictions alone cannot show which actions, feedback, or unmet conditions need inspection. Existing work studies online failure prediction and diagnosis from completed trajectories, yet few methods connect timely warning to a compact symbolic model of execution behavior. We introduce **PrefixGuard**, a neuro-symbolic method that uses learned events to connect online failure warning with finite-state execution diagnosis. A gated recurrent unit (GRU) predicts horizon-specific failure risk from these events. A deterministic finite automaton (DFA) organizes their histories into compact risk-labeled paths for replay and rule-monitor composition. Across all 24 benchmark–horizon settings, PrefixGuard exceeds the strongest evaluated baseline in mean test area under the precision–recall curve (AUPRC). Average gains range from 12.0 to 21.9 percentage points. At a nominal 10% FAR budget, averaged test recall spans 56.1%–93.0% across benchmarks. On one frozen τ^2-Bench DFA, model checking five specifications identifies violations in 21 of 265 benchmark-successful runs.

1 INTRODUCTION

LLM agents carry out multi-step tasks in software engineering, browser navigation, and tool-mediated workflows [31, 34, 4]. Delayed recognition of problems in these workflows can waste resources and allow errors to propagate through subsequent actions. Like an aircraft's black box, recorded traces and terminal verdicts support posthoc analysis, while a cockpit alarm instead warns when there is still time to act (Figure 1a). We ask whether these records can train an agent's cockpit alarm: a monitor that warns from the execution prefix alone. Effective monitoring should identify failure risk before execution ends, giving an operator time to intervene. After an alert, it should help locate the actions, feedback, and unmet task conditions that warrant inspection.

Meeting these needs poses two challenges. **(1) Learning prefix risks from terminal outcomes.** LLM-agent traces mix natural-language messages, tool calls, and environment feedback, while supervision often only consists of a final success or failure label. The challenge is how to learn which patterns in these varied execution histories signal failure risk before the run ends. **(2) Diagnosing execution behavior and risk.** Understanding success and failure requires examining how an agent's behavior advances the task or leaves necessary conditions unresolved. For example, the trajectories in Figure 1b share an initial execution prefix, then diverge and reach different outcomes. The challenge is how to capture execution behavior in a compact symbolic model that supports diagnosis through inspection of the recorded actions, feedback, and task requirements.

Research on process monitoring and formal methods offers complementary tools for prediction and inspection. Neural predictive monitors learn to forecast how incomplete executions will continue and end [26, 25]. Automata extraction can represent recurrent behavior through explicit state transitions [29], and runtime verification can check executions against specified requirements [18]. Work on LLM agents has begun to study online failure prediction and error localization [32], as well as failure diagnosis from completed trajectories [8]. Yet few methods connect online failure prediction with a compact symbolic model for inspecting execution behavior and task requirements. The gap is how to learn both capabilities from agent traces and terminal outcomes while retaining the behavioral distinctions needed for diagnosis.

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.

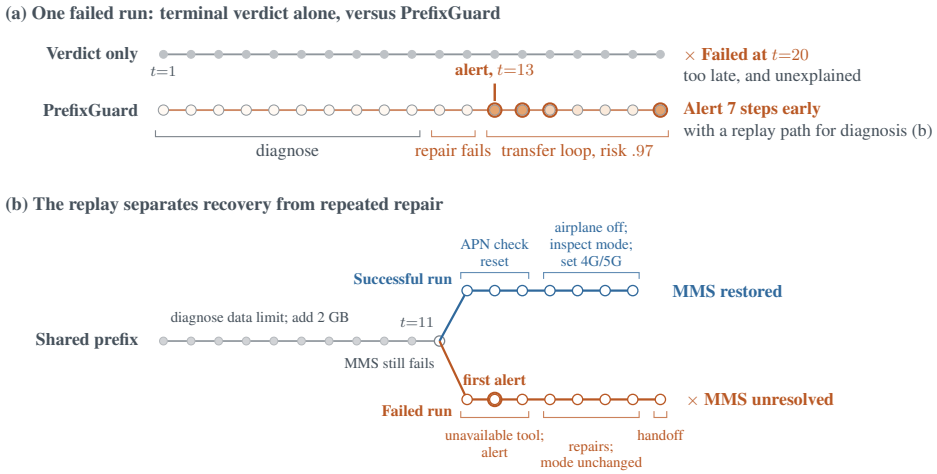

Figure 1: **Warn early, then explain the execution.** (a) A terminal verdict reveals failure only at t=20, whereas PrefixGuard first alerts at t=13 and retains a replayable path. (b) A replay of two same-task runs: after the shared diagnosis and data refill, the successful run checks APN and network mode before restoring MMS; the failed run calls an unavailable tool, repeats repairs without changing the network mode, and hands off unresolved. Figure 5 expands this schematic on the learned DFA.

To address these challenges, we introduce **PrefixGuard**, a neuro-symbolic method that uses learned events to connect online failure warning with finite-state execution diagnosis (Figure 2). Each agent step is converted into a structured action-and-feedback record, then mapped to a small set of event categories. We derive targets for intermediate prefixes from terminal outcomes and jointly train this event mapping with a *gated recurrent unit (GRU)*. During execution, the GRU uses its fixed-size recurrent state to update failure risk within a chosen warning horizon after each step. From the same events and prefix targets, we induce a compact *deterministic finite automaton (DFA)* whose explicit states group histories with compatible risks. The DFA replays an execution as a state path and can be combined with rule monitors to check task requirements. The shared events connect the neural warning to an inspectable symbolic account of execution behavior.

We evaluate warning quality and the DFA's diagnostic capabilities through three questions. **RQ1:** Can terminal-outcome supervision support timely failure warnings (Section 3.2)? **RQ2:** Can compact DFAs retain predictive performance and support execution analysis (Section 3.3)? **RQ3:** Can a learned DFA support task-grounded execution diagnosis through model checking (Section 4)?

Across all 24 benchmark–horizon settings, PrefixGuard exceeds the strongest baseline in mean test AUPRC; benchmark gains average 12.0–21.9 percentage points over horizons. At a nominal 10% FAR budget, horizon-averaged test recall spans 56.1%–93.0% across benchmarks. The same learned events yield DFAs averaging 5.0–32.3 states that retain 90.9% of paired GRU test AUPRC on average. PrefixGuard DFA replay locates the failed run's first alert and shows how different actions after a shared prefix lead to different outcomes. We conduct model checking on the learned DFA for five retail tool-use rules and validate flagged violations against the original traces. Across 359 τ^2-Bench retail executions, we identify violations in 21 of 265 benchmark-successful runs. Together, these results show that terminal outcomes can train a monitor to flag risky prefixes, while the same learned events yield a compact DFA for distinguishing divergent behavior and checking task requirements.

Our contributions are summarized as follows:

- We formulate online agent monitoring around two linked goals: warning of failure before termination and providing a replayable execution path for diagnosis.
- We introduce PrefixGuard, a neuro-symbolic method that learns events for recurrent failure prediction and uses those events to induce a compact, risk-labeled DFA.
- We evaluate PrefixGuard on four agent benchmarks, measuring warning timeliness, false alarms, and how much predictive performance the learned DFAs retain.
- We show how model checking a learned PrefixGuard DFA can locate agents that violate explicit task requirements, with concrete cases validated against the original traces.

2 PREFIXGUARD

2.1 PROBLEM FORMULATION

We formulate online agent monitoring as two connected tasks: predicting failure risk from execution prefixes and representing those prefixes as paths in a risk-labeled DFA for diagnosis.

Observed execution. For task q, an agent produces actions a_t and receives observations o_t from the environment. Each step is $c_t = (a_t, o_t)$, and the observed prefix is $\tau_{:t} = (q, c_1, \ldots, c_t)$. The trajectory $\tau = \tau_{:T}$ ends with verifier outcome $y \in \{0, 1\}$, where $y = 1$ denotes success. During execution, the monitor has access only to this prefix, not future steps or the terminal verdict.

Online failure warning. Given $\tau_{:t}$ and a chosen *warning horizon* H, the problem is to produce a risk score $r_t = f_H(\tau_{:t}) \in [0, 1]$ estimating the probability that the run ends in failure within H further steps. For example, $H = 3$ asks about failure within three more steps, whereas $H = \infty$ asks about eventual failure regardless of when it occurs. A warning is issued when r_t first reaches a threshold γ. The goal is to warn before failed runs terminate while limiting false alarms on successful runs. The horizon specifies how far ahead to predict, not a guaranteed warning lead.

Finite-state abstraction for diagnosis. We seek a learned event alphabet Σ shared by the predictor and a compact risk-labeled DFA $M = (Q, \Sigma, \delta, s_0, \rho)$. Each observed step c_t receives an event symbol $z_t \in \Sigma$; the symbols group steps for failure prediction rather than naming tool operations, so different actions may share one symbol. Here Q is the state set, s_0 its start state, $\delta : Q \times \Sigma \to Q$ the transition function, and $\rho : Q \to [0, 1]$ the horizon-specific state risk. Replaying the events gives $s_t = \delta(s_{t-1}, z_t)$ and a state path $s_0, \ldots, s_t$ to compare with recorded actions and feedback. For example, replaying the learned events shows a shared initial path followed by a failed path that reaches alert state s_6 at t=13 and returns there after further repair attempts, whereas the successful path reaches low-risk s_8 (Figure 5). Aligning these paths with the original actions and feedback shows that the failed run leaves the network mode at 2G with MMS unresolved, while the successful run selects 4G/5G and restores MMS.

2.2 LEARNING A RISK MONITOR

Figure 2 summarizes PrefixGuard: a shared event representation connects the recurrent risk predictor to the DFA used for execution inspection.

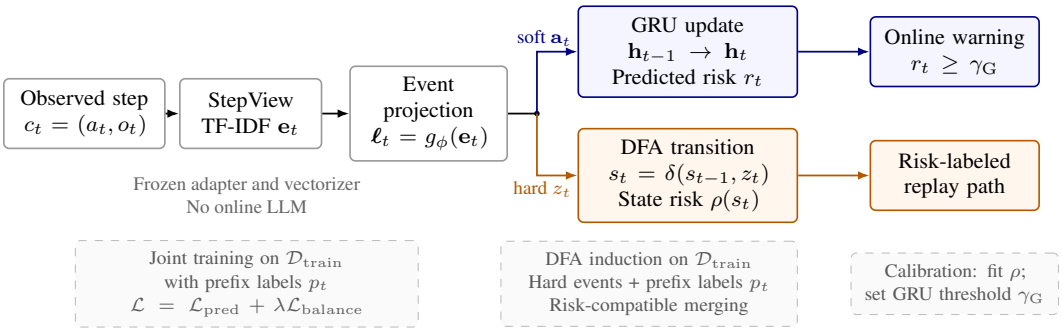

Figure 2: **PrefixGuard architecture.** A shared projection supplies soft events for GRU warnings and hard events for DFA replay. DFA paths are inspected alongside recorded actions and feedback. Bottom annotations summarize training, induction from prefix labels, and calibration.

Step representation. For each execution step, we extract metadata, observation, action, tool, arguments, result, and status into a STEPVIEW record. An LLM proposes the field-extraction parser from training trajectories and we fix it before training the monitor, so no LLM is called at scoring time (Appendix A.2). We encode each record as a TF-IDF [23] vector $\mathbf{e}_t$ of weighted term frequencies, with the vectorizer fitted on training data and kept fixed thereafter.

Risk-aligned event learning. A projection g_ϕ produces unnormalized event scores (logits) $\boldsymbol{\ell}_t = g_\phi(\mathbf{e}_t)$ over $K = |\Sigma|$ discrete event labels. These labels have no predefined tool or action meaning; we interpret them by aligning their occurrences with recorded steps (Section 3.3). At prediction time, the GRU receives *soft* event probabilities $\mathbf{a}_t$, while DFA replay uses the most likely, or *hard*, event z_t:

$$\mathbf{a}_t = \text{softmax}(\boldsymbol{\ell}_t / T_g), \qquad z_t = \arg\max_k \ell_{t,k}. \tag{1}$$

The temperature T_{g} controls how concentrated the event probabilities are; lower values give sharper assignments. During training, we replace $\mathbf{a}_t$ with Gumbel-softmax samples $\widetilde{\mathbf{a}}_t$ at the same temperature, adding noise while retaining differentiability [16]. Prediction-loss gradients flow through the GRU into the projection, making the event partition *risk-aligned*.

Recurrent prediction. The GRU [9] updates $\mathbf{h}_t = \mathrm{GRU}(P(\mathbf{a}_t), \mathbf{h}_{t-1})$ and predicts $r_t = \sigma(\mathbf{w}^\top \mathbf{h}_t + b)$, with $\mathbf{h}_0 = \mathbf{0}$. Here $\mathbf{h}_t$ summarizes the observed event history, P projects event probabilities into the GRU input space, $\mathbf{w}$ and b are the readout weights and bias, and σ is the sigmoid function; all model parameters are learned jointly with g_ϕ. The recurrent state allows the same event to imply different risks after different histories, at a cost that stays constant in t.

Training objective. From completed training trajectories $\mathcal{D}_{\mathrm{train}}$ and their terminal outcomes, we derive horizon-specific targets p_t without step-level annotations (Appendix B.3). The event projection and GRU minimize binary cross-entropy against these targets:

$$\mathcal{L}_{\mathrm{pred}} = \mathbb{E}_{(\tau,y)\sim\mathcal{D}_{\mathrm{train}}}\left[\frac{1}{T}\sum_{t=1}^{T}\mathcal{L}_{\mathrm{BCE}}(r_t, p_t)\right]. \tag{2}$$

Averaging within trajectories weights them equally regardless of length. To encourage confident assignments while keeping multiple event labels in use, we add a balance loss and optimize $\mathcal{L} = \mathcal{L}_{\mathrm{pred}} + \lambda\mathcal{L}_{\mathrm{balance}}$ with $\lambda = 0.1$, where

$$\mathcal{L}_{\mathrm{balance}} = \mathbb{E}_t[\mathcal{H}(\widetilde{\mathbf{a}}_t)] - \beta\mathcal{H}(\mathbb{E}_t[\widetilde{\mathbf{a}}_t]). \tag{3}$$

Entropy $\mathcal{H}$ is taken over non-padding minibatch steps, with $\beta = 1$: the first term penalizes a diffuse assignment at a single step, the second rewards variety across the batch. Validation AUPRC selects the checkpoint, which is frozen before calibration and finite-state induction.

2.3 Learning a Finite-State Model for Diagnosis

We build the DFA of Section 2.1 from the frozen monitor's hard events z_t and prefix labels p_t, adapting ALERGIA's state merging [7] from symbol frequencies to prefix risks: build a prefix tree, merge states whose risks pass a Hoeffding test, and complete missing transitions (Algorithm 1).

Algorithm 1 Risk-compatible DFA induction from labeled prefixes

Require: Ordered labeled prefixes $\mathcal{P} = [(w_i, p_i)]_{i=1}^{N}$ over Σ, with $p_i \in \{0,1\}$; $\eta \in (0,1)$
Build prefix tree in input order; assign node indices on creation and count (m_s, n_s) at endpoints
Let $\widehat{r}_s = m_s/n_s$ if $n_s > 0$, else 0
$R \leftarrow \{s_0\}$; $B \leftarrow \textsc{Candidates}(R)$ ▷ successors of R outside R
while $B \neq \emptyset$ **do**
  $v \leftarrow \arg\min_{v\in B}(-n_v, v)$ ▷ largest support, then smallest index
  $C \leftarrow \{u \in R : \textsc{Compatible}(u, v, \eta)\}$
  if $C = \emptyset$ **then**
    $R \leftarrow R \cup \{v\}$
  else
    $u \leftarrow \arg\min_{u\in C}(|\widehat{r}_u - \widehat{r}_v|, -n_u, u)$
    Redirect the edge from R into v to u; $\textsc{Fold}(u, v)$
  end if
  $B \leftarrow \textsc{Candidates}(R)$
end while
Compact states reachable from s_0; add an absorbing sink for all missing transitions
return (Q, Σ, δ, s_0)

Prefix tree. We first organize the hard event sequences z_t into a prefix tree, where sequences with a shared prefix share the same path. At each node, n counts prefixes ending there (its *support*) and m those with p_t=1; $\widehat{r}$=m/n is the fraction that fail within the horizon. Keeping every prefix would make the DFA large, so we merge prefixes with similar risks.

Compatibility. Two prefixes should share a state only if they have similar risk and the same next event leads to similar risks from both, because a merged state would otherwise erase differences that matter for diagnosis. The compatibility test therefore checks nodes recursively under each shared event, comparing empirical risks with $|\widehat{r}_u - \widehat{r}_v| \le \sqrt{\frac{1}{2}\log\frac{2}{\eta}}\left(n_u^{-1/2} + n_v^{-1/2}\right)$, where n_u, n_v

are the prefix counts at each node [15, 7]. Larger n makes the allowed difference smaller, so two prefixes that appear in many runs are merged only when their risks are close (merge tolerance η=0.05; Appendix A.4). Nodes with no support pass trivially.

Merging. Starting from the root s_0 as the sole DFA state, each round picks the tree node v with the largest support among successors of current DFA states that have not yet been processed. If no current DFA state is compatible, v becomes a new state; otherwise it is folded into the closest-risk compatible state u: the incoming edge is redirected to u, counts are summed, and successors of v are merged into or grafted onto u. When the edge starts at u itself, the fold creates a self-loop. After all nodes are processed, unreachable states are dropped and every missing transition leads to a default state that loops to itself on all events. Ties throughout are broken by support, then node index.

State risks and replay. The training-prefix risks $\widehat{r}$ guide merging only; the final state risks $\rho(s)$ are estimated separately from calibration prefixes, with smoothing as described in Appendix A.4. Replaying an execution yields a risk-labeled state path. To diagnose a run, we compare this path with the recorded actions, feedback, and task requirements (Section 4).

3 Experiments

3.1 Experimental Setup

Datasets. We evaluate PrefixGuard on traces from four benchmarks: WebArena [34], τ^2-Bench [4], SkillsBench [19], and TerminalBench [20], covering 17 distinct agent implementations and 39 backbone models in total. Each benchmark's trajectories are split 70/10/10/10 into train, calibration, validation, and test: train fits the encoder and monitor, calibration sets alert thresholds and DFA state risks, validation selects checkpoints, and test is used only for evaluation (Appendix B).

Baselines. We compare against a predictive-process-monitoring activity LSTM (PPM-LSTM) [26, 25] and three log-anomaly detectors: PCA on activity counts with squared prediction error (PCA (SPE)) [30], DeepLog [11], and LogBERT [13]; figures label PrefixGuard's GRU monitor PG-GRU. We also include a zero-shot DeepSeek-V4-Pro judge scoring a sample of 200 H=3 test prefixes as a reference point (Appendix C.4).

The DFA is compared with its paired GRU on the same test prefixes (Section 3.3), with two induction methods on the same frozen events, an unmerged prefix tree and RPNI [21] (Appendix D.2), and with a differentiable DFA learned end to end under state budgets $Q \in \{8, 16, 32, 64\}$ (Appendix D.3).

Metrics. We measure the online monitor four ways on every benchmark and horizon: AUPRC, how well its risk scores rank failing prefixes above safe ones; recall, the share of failed runs it alerts on; lead, how early the first alert arrives; and false-alarm rate (FAR), the share of successful runs it alerts on. An alert fires when the monitor's score first exceeds a threshold γ_{G}. We set γ_{G} by split-conformal calibration [2] on successful calibration runs at a nominal FAR budget of 10% (Appendix B.4).

We report each DFA's AUPRC alongside its paired GRU, and the number of states. For alert fidelity, we select DFA alert states on calibration runs to match GRU alerts, then report $\Pr(a_D{=}1 \mid a_G{=}1)$ and $\Pr(a_G{=}1 \mid a_D{=}1)$ and their harmonic mean as fidelity F1 (Appendix E.1). Unless noted otherwise, results report means across 3 seeds on test splits, with one population standard deviation.

3.2 Online Failure Warning

RQ1: Can terminal-outcome supervision support timely failure warnings?

RQ1. Terminal-outcome supervision suffices for timely warnings: PrefixGuard exceeds every baseline across all 24 settings, and H trades lead against recall.

PrefixGuard outranks all baselines. PrefixGuard achieved higher mean AUPRC than PPM-LSTM and the log-anomaly baselines in all 24 benchmark–horizon settings (Figure 3, top). Averaged over horizons, the gains over the strongest baseline, PPM-LSTM in every benchmark, range from 12.0 percentage points on TerminalBench to 21.9 on WebArena.

The warning horizon sets a lead–coverage trade-off. Larger H consistently increased lead but did not uniformly raise recall (Figure 3, middle and bottom). A wider horizon labels more prefixes as at-risk, including ones still far from the eventual failure. These noisier targets make the threshold harder to calibrate, so recall can drop. This happened on SkillsBench across all finite horizons and on τ^2-Bench at the longest ones, whereas WebArena recall rose gradually. The $H = \infty$ models offered

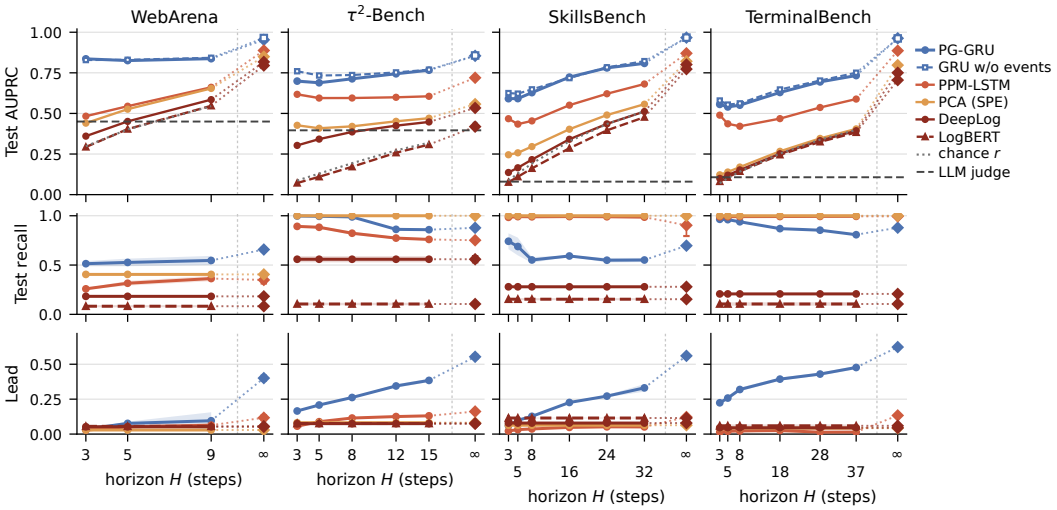

Figure 3: **Warning across horizons.** Test AUPRC (top), recall (middle), and lead (bottom) for PrefixGuard and four baselines; hollow squares: a GRU on the same StepView features without the event projection (Table 1). PPM-LSTM uses `tool|status` activities; DeepLog and LogBERT are scored by mean NLL. Each uses a 10% calibration FAR threshold; realized test FAR is in Appendix C.1. Gray dotted: test prevalence; gray dashed: $H{=}3$ LLM-judge reference.

the greatest lead but predicted eventual rather than imminent failure, making H an operating choice that trades lead for coverage.

Baselines provide little usable advance warning. Under the same 10% calibration FAR budget, PPM-LSTM and PCA catch most failed runs on τ^2-Bench, SkillsBench, and TerminalBench (horizon-averaged recall .81–1.00), while DeepLog and LogBERT catch .08–.56 (Figure 3, middle). Their alerts, however, arrive just before termination: every baseline's horizon-averaged lead is .03–.12, against .15–.39 for PrefixGuard on all four benchmarks (Figure 3, bottom). Realized test FAR is tabulated in Appendices C.1 and C.3; the split-conformal budget bounds it only in expectation.

Zero-shot LLM judging falls well short. DeepSeek-V4-Pro reaches $H{=}3$ AUPRC of .450, .396, .080, and .107 on the four benchmarks (Figure 3, dashed), no better than PCA and far below PrefixGuard, while costing one LLM call per step.

Table 1: **Event ablations.** Mean test ΔAUPRC (pp) over horizons and three seeds: event GRU minus pooled StepView MLP without sequential state; event GRU minus direct-input GRU.

Benchmark	History	Event vs. cont.
WebArena	+2.44	-0.36
τ^2-Bench	-2.37	-2.38
SkillsBench	+0.51	-1.37
TerminalBench	-1.78	-1.26

Where do the gains come from? Table 1 separates two ingredients (Appendix D.1). Sequential state helps at short horizons: against a pooled StepView MLP without it, the recurrent monitor gains 2.44 and 0.51 points on WebArena and SkillsBench but trails by 2.37 and 1.78 on τ^2-Bench and TerminalBench. The event projection is not the source of the margin: a GRU fed the same TF-IDF features without it scores 0.4–2.4 points higher still (Figure 3, hollow squares), 13.3–22.3 above PPM-LSTM. The gain over baselines thus comes from outcome-supervised recurrence over full step records rather than activity codes, and the projection spends a small part of it on the discrete histories that the DFA tested next requires.

3.3 Finite-State Fidelity and Execution Diagnosis

RQ2: Can compact DFAs retain predictive performance and support execution analysis?

RQ2. Compact DFAs retain over 90% of GRU AUPRC while providing interpretable state paths for execution diagnosis.

The DFA preserves most of the GRU's risk ranking. Across benchmarks and horizons, DFAs averaged 5.0–32.3 states and retained 90.9% of their paired GRU's AUPRC on average (Figure 4, top two rows). Alert fidelity (bottom row) is high: with GRU alerts as reference (orange), DFA recall is

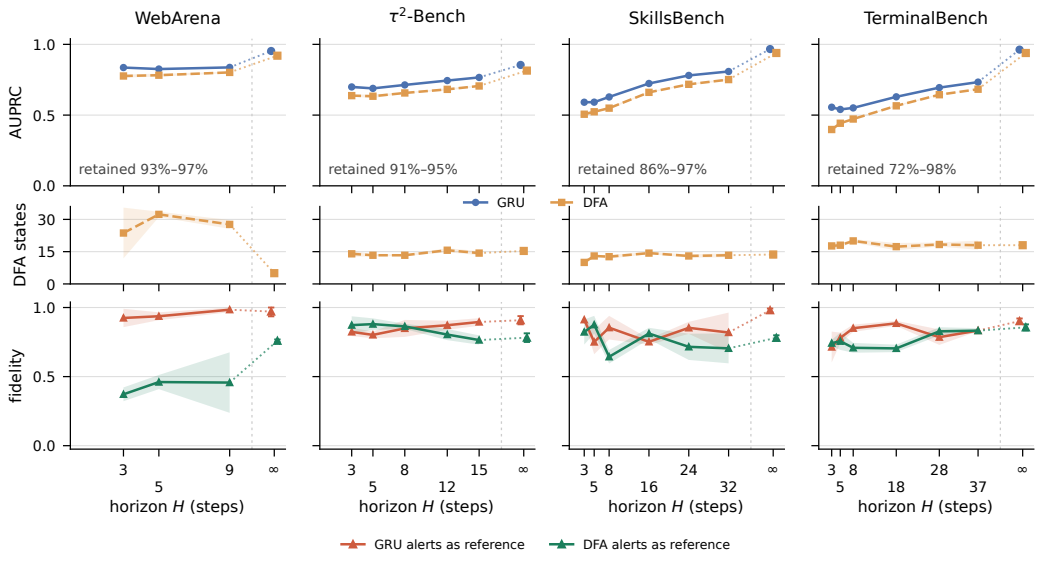

Figure 4: **Finite-state audit across horizons. Rows:** test AUPRC of the GRU and its paired DFA (insets: DFA/GRU retention); DFA state count including the sink; prefix-level alert recall (orange) and precision (green) of the DFA relative to the GRU. Alert states are matched on calibration data without a FAR constraint.

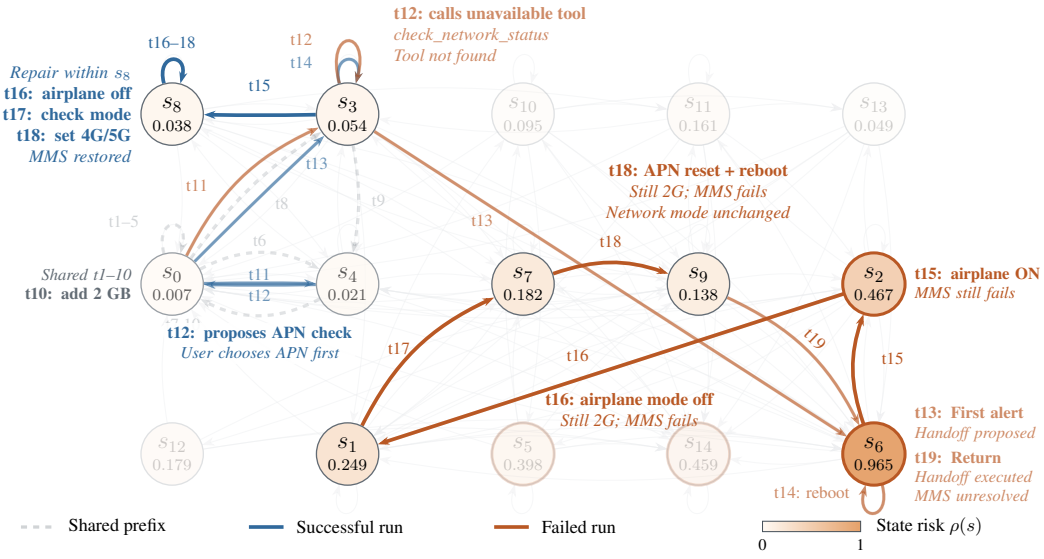

Figure 5: **Risk-labeled DFA paths make warnings inspectable.** In the failed run, s_6 marks the first alert and a later return; the successful run stays in low-risk s_8 during recovery. Node fill shows calibrated H=3 risk; orange rings mark alert states, and unused transitions are faded.

71–98%; with DFA alerts as reference (green), precision is 64–88% on τ^2-Bench, SkillsBench, and TerminalBench. On WebArena precision drops to 37–76%: its few test prefixes leave each DFA state too coarse for the GRU's continuous threshold (Appendix E.2).

The DFA provides interpretability. We replay agent traces through the event projection and DFA to obtain a state path, then align the path with the recorded actions and feedback. Figure 5 illustrates an example of a DFA graph on two τ^2-Bench runs (H=3, seed 13): the failed run reaches alert state s_6 at t=13 (risk .965), passes through further repair states, and returns to s_6 at t=19, while the successful run stays in low-risk s_8 as it restores MMS. The state path locates where the runs diverge, and the aligned actions and feedback reveal why—the failed run leaves the network mode unchanged.

4 CASE STUDY: TASK-GROUNDED DIAGNOSIS WITH MODEL CHECKING

RQ3: Can a learned DFA support task-grounded execution diagnosis through model checking?

Motivation. Section 3.3 showed that the DFA's coarse states can group prefixes with different risks. We argue that domain knowledge can further distinguish prefixes within the same coarse state—for example, whether the agent verified a change before making another one. To test this, we write a safety specification over abstract event classes, compile it into a monitor automaton, and model-check it against the τ^2-Bench DFA from Section 3.3 via product composition. The specification requires a read-class event between every two write-class events. Table 2 shows the result: three prefixes reaching the same DFA state s_9 are separated by the monitor into ready, awaiting a read, and violated, with calibration risks of 0.098, 0.158, and 0.308.

Table 2: **Specification-based refinement at DFA state s_9.** τ^2-Bench calibration ($H = 3$, seed 13); R/W/O = read/write/other. Risk: fitted 3-step failure probability; DFA-only risk: .138.

Monitor state	Event history	Visits	Risk
Ready	R, O, O, W, R	283	.098
Awaiting a read	O, R, O, W, O	53	.158
Violated	O, W, O, O, W, O	69	.308

Adding this specification raised DFA AUPRC from .457 to .510 (Appendix F.1), confirming that product composition can refine the DFA's coarse risk estimates. However, abstract event classes cannot explain *why* a run fails. Diagnosing specific failures requires task-grounded specifications over concrete tool calls. Below we apply the same composition mechanism with such specifications to audit execution outcomes.

Setup. We reuse the same τ^2-Bench DFA, keeping its events and transitions fixed. The study covers 359 retail executions, including 265 successes and 94 failures across 114 tasks and four agents.

Requirements from tool policies. Inspection of the failures yielded five families covering order status (φ_1), replacement pairs (φ_2), item quantities (φ_3), available same-product variants (φ_4), and refund destinations (φ_5). All definitions were fixed before whole-population counting.

Formal specifications. We encode each requirement as an LTL_f safety property [10] (full notation in Appendix F.2): φ_1 checks that the order status permits the operation (e.g. only cancel a pending order); φ_2 checks that replacement lists match in length and that modifications change the item; φ_3 checks that the requested quantity does not exceed stock; φ_4 checks that the replacement item is an available same-product variant; φ_5 checks that the refund goes to the original payment method or the owner's gift card. All properties except φ_2 require retrieving order history before the action; runs that never trigger a property satisfy it vacuously.

Table 3: **Process verdicts by outcome.** Violated/unknown runs; *Any* counts runs with at least one violation.

Property	Success ($n = 265$)	Failure ($n = 94$)
φ_1	12/0	11/0
φ_2	3/0	4/0
φ_3	0/0	1/0
φ_4	1/8	1/4
φ_5	6/0	3/0
Any	21/6	19/0

Model checking on PrefixGuard DFA. We annotate DFA edges with tool-call facts and compose the DFA with each property's monitor automaton via product composition [3]. Each product state (s, u) pairs the learned risk state s with the monitor state u tracking whether the specification has been satisfied or violated. All five specifications reach violating states on the learned DFA without retraining it. Since the DFA can recombine edges into paths not observed in training, we verify each flagged violation against the original trajectory before counting it.

Results. After verification, the five specifications flag 21 of 265 successes (7.92%) and 19 of 94 failures (Table 3). Five flagged successes were never rejected by the runtime yet violate a specification, revealing gaps between the tool's enforcement and the stated policy (Appendix F.2).

Case: diagnosing a failed retail task. Now we give an example of how the product graph pinpoints a failure cause. In retail task 27, the customer prefers exchanging boots over returning them if both options cannot be fulfilled. Figure 6 compares a failed and a successful run: the failed agent attempts a return first, which changes the order status and blocks the subsequent exchange; the successful agent completes the exchange first and then handles the return. Both runs violate φ_1 (operating on an order whose status no longer permits the action), but only the successful run passes the database check. The two violations share the same DFA state s_2; the monitor state distinguishes them by tracking the status change that the risk state alone cannot capture. The product graph identifies the blocked precondition, and the task context explains why it matters (Appendix F.3).

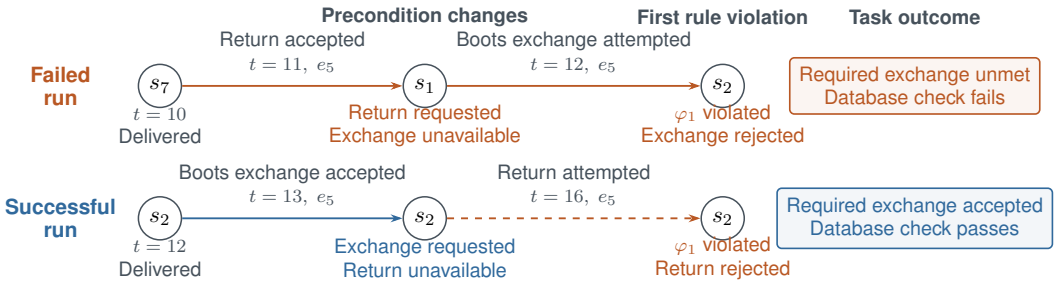

Figure 6: **Task-27 diagnosis on PrefixGuard's DFA.** Both runs violate the order-status requirement φ_1, but only the failed run leaves the preferred exchange unmet. Dashed arrows omit intermediate transitions; task outcomes are annotations, not DFA states.

RQ3. The learned DFA supports task-grounded diagnosis through model checking: product composition with LTL_f specifications flags process violations, refines risk beyond DFA states, and pinpoints failure causes without retraining the DFA.

5 Related Work

Predictive monitoring and alarms. Predictive process monitoring forecasts outcomes and continuations of incomplete executions from event logs, including with LSTM sequence models [26, 25]. Log-anomaly detection instead identifies unusual event sequences using methods such as PCA, next-event prediction, and masked-event modeling [30, 11, 13]. These two lines motivate our matched PPM-LSTM, PCA, DeepLog, and LogBERT baselines. Alarm-based prescriptive monitoring additionally selects when to intervene using explicit costs [12]; PrefixGuard evaluates warnings but does not optimize interventions. LLM-as-judge and digital-agent evaluators score observed behavior or task success [33, 22]. PrefixGuard instead scores every step online without deployment-time LLM calls and evaluates warnings by lead, recall, and realized false alarms under frozen thresholds. Concurrently, AgentForesight [32] trains a 7B auditor with reinforcement learning to localize the decisive error in multi-agent trajectories, targeting step-level attribution rather than failure risk.

Runtime verification, specification mining, and trace abstraction. Runtime verification checks executions against supplied formal properties [18, 5], whereas specification mining infers likely interaction protocols from observed traces [1]. Dialog2Flow clusters utterance embeddings into discrete workflow sequences [6]. PrefixGuard separates these roles: outcome supervision learns a risk-aligned event alphabet and audit graph, while composition with explicit rule monitors checks task requirements.

Finite-state learning and neural-model audit. Weiss et al. [29] extract automata from recurrent networks using the network as an oracle, while Shvo et al. [24] learn finite-state sequence classifiers through discrete optimization. RPNI merges prefix-tree states consistent with labeled examples [21], ALERGIA those with Hoeffding-compatible frequencies [7]; DeepDFA learns a DFA end to end through relaxed transitions [27]. We compare risk-compatible merging with a prefix tree, RPNI, and a differentiable DFA; the paired GRU measures neural–symbolic fidelity. PrefixGuard's graph induction uses failure-labeled prefixes rather than GRU outputs as an oracle.

6 Conclusion and Future Work

PrefixGuard learns a shared event representation from terminal outcomes and uses it for online failure warning and finite-state execution diagnosis. Across 24 benchmark–horizon settings, the event-based GRU outranks all baselines and a zero-shot LLM judge, and the warning horizon trades lead against recall. Compact DFAs induced from the same events retain about 91% of the GRU's AUPRC and expose state paths that replay divergence and support model checking of task-grounded requirements. Performance on unseen tasks and agents remains unmeasured. Controlled intervention studies should test whether correcting identified requirement violations improves task outcomes.

AI USE STATEMENT

LLMs were used to propose the offline StepView adapters described in Appendix A.2. AI assistance was also used to reorganize and edit the manuscript, and format result tables. AI agents also assisted the descriptive replay annotations.

REPRODUCIBILITY STATEMENT

Appendix A specifies the StepView adapters and their frozen induction protocol (Appendices A.1 and A.2), the encoder, event projection, and GRU settings with the training and checkpoint-selection recipe (Appendix A.3, Tables 4 and 5), the risk-compatible induction and audit-readout procedure (Appendix A.4, Algorithm 2), and recorded compute (Appendix A.5). Appendix B documents the trace sources and agent composition, the data splits, the prefix-label construction for every horizon, and the split-conformal calibration and metric definitions. Appendices C–E report per-horizon results with seed variation for PrefixGuard, every baseline, the ablations, and the DFA audits, and Appendix C.4 reproduces the LLM-judge prompts. Appendix F gives the formal specifications, monitor constructions, and independent FLLOAT checks behind the model-checking results. Source code, the frozen data splits, and the evaluator exports that generate every table and figure will be released with the paper.

REFERENCES

[1] G. Ammons, R. Bodík, and J. R. Larus. Mining specifications. *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2002.

[2] A. N. Angelopoulos and S. Bates. Conformal Prediction: A Gentle Introduction. *Foundations and Trends in Machine Learning*, 2023.

[3] C. Baier and J.-P. Katoen. Principles of Model Checking. *MIT Press*, 2008.

[4] V. Barres, H. Dong, S. Ray, X. Si, and K. Narasimhan. τ^2-Bench: Evaluating Conversational Agents in a Dual-Control Environment. *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

[5] A. Bauer, M. Leucker, and C. Schallhart. Runtime Verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology*, 2011.

[6] S. Burdisso, S. Madikeri, and P. Motlicek. Dialog2Flow: Pre-training Soft-Contrastive Action-Driven Sentence Embeddings for Automatic Dialog Flow Extraction. *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.

[7] R. C. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. *Grammatical Inference and Applications, Second International Colloquium (ICGI-94)*, 1994.

[8] M. Cemri, M. Z. Pan, S. Yang, L. A. Agrawal, B. Chopra, et al. Why Do Multi-Agent LLM Systems Fail? *Advances in Neural Information Processing Systems*, 2025.

[9] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, et al. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[10] G. De Giacomo and M. Y. Vardi. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[11] M. Du, F. Li, G. Zheng, and V. Srikumar. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017.

[12] S. A. Fahrenkrog-Petersen, N. Tax, I. Teinemaa, M. Dumas, M. de Leoni, et al. Fire now, fire later: alarm-based systems for prescriptive process monitoring. *Knowledge and Information Systems*, 2022.

[13] H. Guo, S. Yuan, and X. Wu. LogBERT: Log Anomaly Detection via BERT. *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021.

[14] H. Hansson and B. Jonsson. A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computing*, 1994.

[15] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 1963.

[16] E. Jang, S. Gu, and B. Poole. Categorical Reparameterization with Gumbel-Softmax. *International Conference on Learning Representations (ICLR)*, 2017.

[17] O. Kupferman and M. Y. Vardi. Model Checking of Safety Properties. *Formal Methods in System Design*, 2001.

[18] M. Leucker and C. Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 2009.

[19] X. Li, Y. Liu, W. Chen, B. You, Z. Di, et al. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. *arXiv:2602.12670*, 2026.

[20] M. A. Merrill, A. G. Shaw, N. Carlini, B. Li, H. Raj, et al. Terminal-Bench: Benchmarking Agents on Hard, Realistic Tasks in Command Line Interfaces. *The Fourteenth International Conference on Learning Representations*, 2026.

[21] J. Oncina and P. García. Inferring regular languages in polynomial updated time. *Pattern Recognition and Image Analysis*, 1992.

[22] J. Pan, Y. Zhang, N. Tomlin, Y. Zhou, S. Levine, et al. Autonomous Evaluation and Refinement of Digital Agents. *Proceedings of the First Conference on Language Modeling (COLM)*, 2024.

[23] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 1988.

[24] M. Shvo, A. C. Li, R. Toro Icarte, and S. A. McIlraith. Interpretable Sequence Classification via Discrete Optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[25] N. Tax, I. Verenich, M. La Rosa, and M. Dumas. Predictive Business Process Monitoring with LSTM Neural Networks. *Advanced Information Systems Engineering (CAiSE)*, 2017.

[26] I. Teinemaa, M. Dumas, M. La Rosa, and F. M. Maggi. Outcome-Oriented Predictive Process Monitoring: Review and Benchmark. *ACM Transactions on Knowledge Discovery from Data*, 2019.

[27] E. Umili and R. Capobianco. DeepDFA: Automata Learning through Neural Probabilistic Relaxations. *ECAI 2024*, 2024.

[28] B. von Berg and B. K. Aichernig. Extending AALpy with Passive Learning: A Generalized State-Merging Approach. *Computer Aided Verification (CAV 2025)*, 2025.

[29] G. Weiss, Y. Goldberg, and E. Yahav. Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples. *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, 2018.

[30] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan. Detecting Large-Scale System Problems by Mining Console Logs. *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP)*, 2009.

[31] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, et al. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. *Advances in Neural Information Processing Systems*, 2024.

[32] B. Zhang, J. Zhu, Z. Shi, D. Liu, and R. Tang. AgentForesight: Online Auditing for Early Failure Prediction in Multi-Agent Systems. 2026.

[33] L. Zheng, W. Chiang, Y. Sheng, S. Zhuang, Z. Wu, et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. *Advances in Neural Information Processing Systems 36: NeurIPS 2023*, 2023.

[34] S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, et al. WebArena: A Realistic Web Environment for Building Autonomous Agents. *The Twelfth International Conference on Learning Representations, ICLR 2024*, 2024.

APPENDIX CONTENTS

A Model and Training Details **14**
A.1 StepView: LLM-Induced Adapter, Frozen Before Training . . . 14
A.2 Offline Adapter Induction . . . 14
A.3 Encoding, Training, and Checkpoint Selection . . . 14
A.4 Risk-Aligned Induction and Audit Readouts . . . 15
A.5 Computational Setup . . . 16

B Data and Evaluation Protocol **17**
B.1 Trace Sources and Benchmark Composition . . . 17
B.2 Data Splits . . . 17
B.3 Prefix Labels and Input Availability . . . 18
B.4 Calibration and Metric Definitions . . . 18

C Extended Warning Results **19**
C.1 Full Horizon Results and Seed Variation . . . 19
C.2 Baselines on the Same Data Splits . . . 19
C.3 Baseline Alert Operating Points . . . 20
C.4 Zero-Shot LLM Judge Reference . . . 21

D Design Analysis **23**
D.1 Event Representation and Predictor Ablations . . . 23
D.2 The Choice of DFA Induction Method . . . 24
D.3 Directly Learned DFAs . . . 24
D.4 Where Finite-State Fidelity Breaks . . . 26
D.5 Merge-Parameter Sensitivity . . . 27

E Finite-State Audit and Replay Evidence **28**
E.1 Matched-Alert-Set Fidelity . . . 28
E.2 DFA Risk-Threshold Operating Points . . . 28
E.3 State Support and Replay Coverage . . . 29
E.4 Semantic Review Protocol and Counts . . . 30
E.5 Detailed Cases and Selection Procedure . . . 32
E.6 Boundary Case: A Silent Risk-Thresholded DFA . . . 32

F Rule Refinement and Task-Grounded Verification **33**
F.1 Rule Refinement with Abstract Events . . . 33
F.2 Process-Conformance Audit . . . 34
F.3 Task-27 Temporal Ordering Example . . . 35
F.4 Task-Grounded Requirements and Counterexample Validation . . . 36

A Model and Training Details

A.1 StepView: LLM-Induced Adapter, Frozen Before Training

StepView records use benchmark-specific field mappings proposed once by an offline LLM and fixed before monitor training. Each record contains metadata, a reduced observation, action text, tool name, serialized arguments, result text, and status. The serializer collapses whitespace in text and encodes arguments as JSON with sorted keys and ASCII escaping. Missing tool names or statuses become `NONE`; missing metadata becomes `NO_METADATA`; an empty reduced observation renders as `NONE`. Empty action and result strings remain empty. Unrecognized tool names remain text features rather than triggering a separate full-step fallback.

The renderer retains at most eight observation units: dialogue turns for τ^2-Bench, log blocks for TerminalBench, trace-family-specific units for SkillsBench, and lexical lines for WebArena. The dataset identity determines the serializer. For WebArena, the encoded text contains metadata, action, tool, arguments, result, and status. For the other three benchmarks, it also includes the reduced observation (Table 4). Metadata appears within the serialized text and is not supplied through a separate feature channel.

Table 4: Input and model settings. K is the event-alphabet size and d_h is the GRU hidden dimension. The event-projection hidden layer has width 128 in every corpus.

Corpus	K	d_h
WebArena	16	16
τ^2-Bench	16	16
SkillsBench	16	16
TerminalBench	32	32

A.2 Offline Adapter Induction

For each benchmark, an LLM proposes how to map the training-trace fields into StepView; the accepted mapping is fixed before model training and applied to all splits. The LLM runs only offline: GRU scoring and DFA replay use the fixed mapping, which is chosen independently of outcome labels.

For each benchmark, twelve raw steps are selected from the first 64 training trajectories: four initial, four middle, two tool/action, and two anomalous steps. Within each group, examples are ordered by `sha256(trajectory_id:step_index)`, making the selection independent of labels and model scores. The proposal uses `gpt-5.4-nano` at temperature zero with a constrained JSON schema. The prompt names the seven StepView fields, asks for direct extraction rather than semantic rewriting, requires unavailable fields to remain unknown, and restricts selections to supported source fields and observation-reduction rules. The proposed mapping is accepted only after schema and field-validity checks, then fixed before training.

A.3 Encoding, Training, and Checkpoint Selection

Encoding. Each serialized step is transformed by a unigram/bigram TF-IDF vectorizer fitted on training text and held fixed afterward. All runs use the cuML implementation with 4,096 input dimensions. Step text is encoded without character truncation.

For cuML vocabulary fitting, the implementation guards against 32-bit string-offset overflow. If 3.5 times the total input character count exceeds 1.5×10^9, it uniformly subsamples training texts with a separate NumPy seed of 42 to meet that estimated budget. This affects vocabulary fitting only; all steps are transformed afterward. The fitted vocabulary and TF-IDF weights are held fixed during evaluation.

Model. For a step vector $\mathbf{e}_t \in \mathbb{R}^{4096}$, the event projection is

$$\boldsymbol{\ell}_t = W_2\,\mathrm{GELU}(W_1\mathbf{e}_t + b_1) + b_2, \qquad W_1 \in \mathbb{R}^{128\times 4096}, \quad W_2 \in \mathbb{R}^{K\times 128}.$$

Training uses soft Gumbel–Softmax assignments without a straight-through estimator. The GRU receives the event probabilities through a learned linear projection and GELU, updates a single recurrent layer of width d_h, and applies a linear sigmoid risk head. Table 4 gives K and d_h. At evaluation, Gumbel noise is removed:

$$\mathbf{a}_t = \mathrm{softmax}(\boldsymbol{\ell}_t/T_{\mathrm{g}}), \qquad z_t = \arg\max_k \ell_{t,k}.$$

The GRU consumes $\mathbf{a}_t$, whereas DFA induction and replay consume z_t. The evaluation temperature T_{g} is restored from the selected checkpoint.

Table 5: Shared training settings. The prediction loss uses no class weights.

Setting	Value
Optimizer	AdamW
Learning rate / weight decay	10^{-3} / 10^{-4}
Epochs / batch size	24 / 64 trajectories
Prediction / balance weights	1 / 0.1
Marginal-entropy weight	1
Gumbel temperature	Linear, 1.0 at epoch 1 to 0.25 at epoch 24
Sequence handling	Full trajectories; padded batches and label masks
Batch order	Input order; length bucketing disabled
Gradient clipping	None in the evaluated training path
Checkpoint selection	Validation AUPRC, with tie rules below

Loss and batching. For batch $\mathcal{B}$, let V_i be the valid labeled positions of trajectory i. The loss is

$$\mathcal{L}_{\mathrm{pred}} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \frac{1}{|V_i|} \sum_{t \in V_i} \mathrm{BCE}(r_{i,t}, p^H_{i,t}), \tag{4}$$

$$\mathcal{L}_{\mathrm{balance}} = \mathbb{E}_{i,t}[\mathcal{H}(\widetilde{\mathbf{a}}_{i,t})] - \mathcal{H}(\mathbb{E}_{i,t}[\widetilde{\mathbf{a}}_{i,t}]), \qquad \mathcal{L} = \mathcal{L}_{\mathrm{pred}} + 0.1\mathcal{L}_{\mathrm{balance}}. \tag{5}$$

The entropy averages cover non-padding steps in the batch. Padding and label masks exclude invalid positions from the prediction loss; full trajectories are retained. The ordinary batch iterator preserves input order, without an additional epoch-wise shuffle. Training seeds initialize Python, NumPy, PyTorch, and CUDA random-number generators.

Selection and restoration. Validation is performed after each epoch. The checkpoint key is lexicographic: higher validation AUPRC, then lower validation expected calibration error (ECE), then higher validation alert lead. An exact tie retains the earlier checkpoint. The linear temperature schedule is $\tau_e = 1 - 0.75(e-1)/23$. Selected epochs range from 9–23 on WebArena, 16–24 on τ^2-Bench, 15–24 on SkillsBench, and 14–24 on TerminalBench. Evaluation uses the selected epoch's temperature, event projection, and GRU weights, together with the same serializer and fitted vectorizer.

A.4 Risk-Aligned Induction and Audit Readouts

Prefix counts and merging. The prefix tree is built from every labeled training prefix and its hard-event sequence. Each occurrence increments the positive and total counts (m_s, n_s) only at its endpoint; repeated histories with conflicting labels remain in the same counts. Supported states have empirical risk $\widehat{r}_s = m_s/n_s$. Two nodes u, v are compatible when

$$|\widehat{r}_u - \widehat{r}_v| \le \sqrt{\tfrac{1}{2}\log(2/\eta)}\left(n_u^{-1/2} + n_v^{-1/2}\right), \qquad \eta = 0.05, \tag{6}$$

and the same test holds recursively for successors under every shared event. If either node has zero support, the local test passes; recursive successor checks still apply. No conflicting-prefix filtering is used by this induction rule.

The retained red set starts with the root. Among its non-red children, the algorithm selects the node with greatest support, breaking ties by the smallest node index. It merges into the compatible red node with closest empirical risk, then greatest support, then smallest index; if none is compatible, the selected node becomes red. Folding sums endpoint counts and recursively combines shared successors. After merging, the implementation compacts reachable nodes and completes absent transitions with an absorbing sink.

State risks. Calibration replay gives a state positive count c_s, support N_s, and global positive-prefix prevalence $\bar{p}_{\mathrm{cal}}$. The risk lookup is

$$\rho(s) = \frac{c_s + 5\bar{p}_{\mathrm{cal}}}{N_s + 5}. \tag{7}$$

This rule also covers unsupported states and the sink, assigning the global prevalence when $N_s = 0$. An own-FAR10 readout applies Eq. 9 to maxima of $\rho(s_t)$ on successful calibration trajectories and freezes the resulting scalar threshold.

Matched alert set. Let $g_{i,t} = \mathbb{I}[r_{i,t} \geq \gamma_G]$ be the frozen GRU decision on calibration. For each visited state, compute $u_s = \sum_{i,t:s_{i,t}=s} g_{i,t}$ and $v_s = \#\{(i,t) : s_{i,t} = s\}$. Sort states by decreasing u_s/v_s, breaking ties by decreasing u_s and then increasing state ID. For the first k states S_k, choose

$$k^* = \arg\max_k F_{1,\mathrm{cal}}\big(\mathbb{I}[s_{i,t} \in S_k], g_{i,t}\big), \qquad a_D(\tau_{:t}) = \mathbb{I}[s_t \in S_{k^*}].$$

The scan retains the smallest k attaining the maximum. The readout is undefined if the GRU has no calibration alerts; states unvisited on calibration are excluded from the selected set. State membership is then frozen for test. This optimizes GRU-relative agreement without a FAR constraint and is distinct from thresholding $\rho(s)$.

Algorithm 2 constructs both DFA readouts after model selection and graph induction. Its inputs are calibration records containing the replayed state $s_{i,t}$, prefix label $p_{i,t}$, and frozen GRU alert $g_{i,t}$. Write $\mathrm{Threshold}_\alpha$ for the successful-trajectory maximum rule in Eq. 9. The symbol $\perp$ denotes an undefined matched-alert readout, rather than an empty alert set.

Algorithm 2 Calibrating DFA risk and GRU-matched alert readouts

Require: Frozen DFA states Q (including sink); nominal FAR budget α
Require: Calibration records $\{(y_i, (s_{i,t}, p_{i,t}, g_{i,t})_{t=1}^{T_i})\}_{i=1}^{n}$
Ensure: State risks ρ, risk threshold γ_D, matched alert states S^*

$\bar{p} \leftarrow \sum_{i,t} p_{i,t} / \sum_i T_i$; $G \leftarrow \sum_{i,t} g_{i,t}$
for each $s \in Q$ **do**
  $I_s \leftarrow \{(i,t) : s_{i,t} = s\}$; $N_s \leftarrow |I_s|$
  $c_s \leftarrow \sum_{(i,t)\in I_s} p_{i,t}$; $u_s \leftarrow \sum_{(i,t)\in I_s} g_{i,t}$
  $\rho(s) \leftarrow (c_s + 5\bar{p})/(N_s + 5)$
end for
$\gamma_D \leftarrow \mathrm{Threshold}_\alpha(\{\max_t \rho(s_{i,t}) : y_i = 1\})$
if $G = 0$ **then**
  return $(\rho, \gamma_D, \perp)$
end if
$(s_1, \ldots, s_m) \leftarrow \mathrm{sort}_{s:N_s>0}(-u_s/N_s, -u_s, s)$
$\mathrm{TP} \leftarrow 0$; $\mathrm{FP} \leftarrow 0$; $F^* \leftarrow -1$; $S^* \leftarrow \emptyset$
for $k = 1, \ldots, m$ **do**
  $\mathrm{TP} \leftarrow \mathrm{TP} + u_{s_k}$; $\mathrm{FP} \leftarrow \mathrm{FP} + N_{s_k} - u_{s_k}$
  $F_k \leftarrow 2\mathrm{TP}/(2\mathrm{TP} + \mathrm{FP} + G - \mathrm{TP})$
  if $F_k > F^*$ **then**
    $F^* \leftarrow F_k$; $S^* \leftarrow \{s_1, \ldots, s_k\}$ ▷ retain smallest tied set
  end if
end for
return (ρ, γ_D, S^*)

A.5 Computational Setup

Training uses dense, padded GRU batches on CUDA, with step embeddings stored in 32-bit floating point and reused across epochs. Input, model, and optimization settings are given in Tables 4 and 5; Algorithm 2 specifies audit-readout calibration after training.

Table 6 summarizes recorded training and induction times. Training time includes data loading, encoding, optimization, validation, checkpoint saving, and scheduling overhead. Induction time covers graph construction and merging, excluding encoding, data loading, readout calibration, and test scoring.

Table 6: Elapsed-time ranges in seconds.

Benchmark	Training (s)	DFA induction (s)
WebArena	121–302	0.07–0.57
τ^2-Bench	180–301	0.36–1.46
SkillsBench	242–302	2.82–3.49
TerminalBench	2,722–7,665	9.13–16.20

B Data and Evaluation Protocol

B.1 Trace Sources and Benchmark Composition

The main evaluation uses frozen execution traces from WebArena, τ^2-Bench, SkillsBench, and TerminalBench. Table 7 describes the frozen corpora, before splitting. An agent implementation denotes the program and execution variant that drives the underlying language model. Model counts pool provider aliases, snapshot dates, and equivalent version spellings; agent counts retain named execution variants while pooling prompt configurations. The union across corpora contains 17 agent implementations and 39 backbone models.

Table 7: Composition of the current trace corpora. Success rates and mean lengths are computed over trajectories; agent and model counts use the normalization rules above.

Corpus	Trajectories	Tasks	Agents	Models	Success (%)	Mean steps
WebArena	6,047	812	3	5	22.4	9.3
τ^2-Bench	10,832	392	4	4	66.3	15.5
SkillsBench	10,951	89	5	7	27.5	32.4
TerminalBench	34,397	89	9	31	32.2	36.9

Trace sources. WebArena combines 4,425 executions from its 2023 official trace releases[1] with 810 ColorBrowserAgent executions[2] using GPT-5 and 812 WebOperator executions[3] using GPT-4o. The τ^2-Bench traces come from the official bundled evaluation results.[4] SkillsBench uses archived trial records[5] containing each trial's agent trace, configuration, and verifier result. TerminalBench uses the public trajectory dataset.[6]

B.2 Data Splits

Trajectory IDs are disjoint across train, calibration, validation, and test; the allocator ensures that every task in a held-out split also occurs in train. Not every task occurs in every held-out split: WebArena has 549, 546, and 549 distinct tasks in calibration, validation, and test, respectively; SkillsBench has 86, 86, and 88. All 392 τ^2-Bench tasks and all 89 TerminalBench tasks occur in each split.

Table 8: Frozen split sizes. Parentheses give the number of successful trajectories. Validation selects the GRU checkpoint; calibration determines the frozen readouts.

Corpus	Train	Calibration	Validation	Test
WebArena	4,233 (949)	605 (136)	605 (136)	604 (135)
τ^2-Bench	7,583 (5,027)	1,083 (718)	1,083 (718)	1,083 (718)
SkillsBench	7,666 (2,107)	1,095 (301)	1,095 (301)	1,095 (301)
TerminalBench	24,077 (7,728)	3,440 (1,111)	3,440 (1,118)	3,440 (1,122)

Allocation. WebArena, τ^2-Bench, and SkillsBench use task-by-outcome integer allocation with target proportions 70/10/10/10. The allocator satisfies global outcome quotas while minimizing absolute deviations from the target group allocations. Within groups, a hash of the split seed and trajectory ID fixes the assignment order; the split seed is 13. The all-split task-coverage constraint applies to tasks with at least eight executions for WebArena and four for the other two corpora. TerminalBench instead shuffles executions within each task with split seed 1, assigns rounded 10% shares to calibration, validation, and test, and gives the remainder to train, while ensuring each task appears in each split. Its allocation is not stratified by outcome.

[1] https://github.com/web-arena-x/webarena/blob/main/resources/README.md

[2] https://github.com/MadeAgents/browser-agent/tree/main/trajectories/webarena

[3] huggingface.co/datasets/mahirlabibdihan/weboperator-experiments

[4] https://github.com/sierra-research/tau2-bench/tree/main/data/tau2/results/final

[5] https://github.com/benchflow-ai/skillsbench-trajectories

[6] https://huggingface.co/datasets/yoonholee/terminalbench-trajectories

B.3 Prefix Labels and Input Availability

Let $y_i = 1$ denote trajectory success, T_i its observed length, and $t \in \{1, \ldots, T_i\}$ the step index after observing the action and its result. For a finite warning horizon $H \in \mathbb{Z}^+$, the prefix target is

$$p_{i,t}^H = \mathbb{I}[y_i = 0 \ \wedge \ T_i - t \leq H]. \tag{8}$$

A failed trajectory contributes at most $H + 1$ positive prefixes, including termination. Earlier prefixes are negative for this horizon, even if they already contain evidence of eventual failure; all prefixes of successful trajectories are negative. For $H = \infty$, every prefix receives the trajectory's failure label $\mathbb{I}[y_i = 0]$. The stored implementation uses $H = 100{,}000$ for this endpoint, exceeding every trajectory length in these corpora. Writing p_t for $p_{i,t}^H$ when the trajectory and horizon are fixed, the target risk is $\mathrm{Risk}_H(\tau_{:t}) = \Pr(p_t = 1 \mid \tau_{:t})$. It concerns failure of the run within the chosen horizon, rather than whether the current action is correct. Neither future steps nor T_i are supplied as an online feature; T_i and the outcome define the training target and retrospective evaluation.

Table 9: Locked-test positive-prefix prevalence r_H for each evaluated horizon. Values are percentages and are identical across training seeds. Test prefix counts are 5,692 / 16,851 / 35,663 / 124,886 in column order.

WebArena		τ^2-Bench		SkillsBench		TerminalBench	
H	r_H	H	r_H	H	r_H	H	r_H
3	29.83	3	8.66	3	8.74	3	7.04
5	40.14	5	12.97	5	13.02	5	10.07
9	54.44	8	19.35	8	19.15	8	14.16
∞	80.46	12	27.25	16	32.85	18	25.30
		15	31.98	24	43.48	28	33.82
		∞	43.43	32	51.27	37	39.89
				∞	76.93	∞	75.95

B.4 Calibration and Metric Definitions

Each benchmark and horizon has a separately trained GRU. The TF-IDF encoder and model weights are fitted using training trajectories; validation AUPRC selects the checkpoint. The selected model is frozen before calibration. For the DFA, training prefixes determine the graph, calibration prefixes determine state risks, and calibration GRU alerts determine the matched alert set. Test trajectories are used only to evaluate the frozen model and readouts.

Frozen FAR threshold. For the n successful calibration trajectories, let $m_i = \max_t r_{i,t}$ and sort these maxima as $m_{(1)} \leq \cdots \leq m_{(n)}$. At nominal budget $\alpha = 0.10$, the implementation sets

$$k = \lceil (n+1)(1-\alpha) \rceil, \qquad \gamma = \begin{cases} \mathrm{nextafter}(m_{(k)}, +\infty), & k \leq n, \\ +\infty, & k > n. \end{cases} \tag{9}$$

Here `nextafter` returns the next representable value above the selected maximum, so the alert test $r \geq \gamma$ excludes calibration ties at that value. All reported cells have successful calibration trajectories (Table 8). For a scorer fixed independently of calibration, the split-conformal interpretation requires exchangeability of successful calibration and future trajectories. Under this assumption, the future successful-run maximum exceeds $m_{(k)}$ with probability at most $(n+1-k)/(n+1) \leq \alpha$ when $k \leq n$. This follows from its exchangeable rank among $n+1$ maxima; excluding ties is conservative, and $k > n$ produces no alerts. Thus $\Pr(A < T \mid y = 1) \leq \Pr(A \leq T \mid y = 1) \leq \alpha$, with probability taken over calibration and the future successful trajectory. The bound is marginal over calibration and future trajectories. The DFA's state risks and its own FAR threshold share calibration data; the same numerical rule therefore supplies an empirical operating point for that readout. The matched alert set optimizes agreement with GRU alerts and is not constrained by FAR.

Ranking and warning. AUPRC is computed as non-interpolated average precision over the pooled labeled prefixes. It weights prefixes equally, whereas the prediction training loss weights trajectories equally. Let $A_i = \min\{t : r_{i,t} \geq \gamma\}$, with $A_i = T_i + 1$ for no alert, and let N_+ and N_- count

successful and failed test trajectories. The warning metrics are

$$\text{FAR} = \frac{1}{N_+} \sum_{i:y_i=1} \mathbb{I}[A_i < T_i], \tag{10}$$

$$\text{Recall} = \frac{1}{N_-} \sum_{i:y_i=0} \mathbb{I}[A_i < T_i], \tag{11}$$

$$\text{Lead} = \frac{1}{N_-} \sum_{i:y_i=0} \frac{(T_i - A_i)_+}{T_i}. \tag{12}$$

C Extended Warning Results

C.1 Full Horizon Results and Seed Variation

Tables 10–13 expand the main warning figure for all 24 benchmark–horizon settings and three training seeds per setting. Validation AUPRC selects checkpoints; test AUPRC uses held-out prefixes. Both prevalences are reported because changing H changes the positive target and its frequency. The deployment columns use the frozen calibration threshold and measure trajectory-level FAR, recall, and lead as defined in Appendix B.4.

Each setting reports the mean and population SD across its three seeds. The final column identifies every seed whose realized test FAR exceeds 0.10, even where the mean remains below the nominal budget.

Table 10: WebArena horizon results. r_v/r_t gives validation/test positive-prefix prevalence. FAR, recall, and lead use locked test.

H	Val AP	Test AP	r_v/r_t	FAR	Recall	Lead	Seeds $>.10$
3	$.881 \pm .003$	$.836 \pm .001$	.299/.298	$.074 \pm .006$	$.514 \pm .026$	$.035 \pm .003$	–
5	$.861 \pm .008$	$.825 \pm .007$	.403/.401	$.072 \pm .007$	$.527 \pm .038$	$.077 \pm .012$	–
9	$.858 \pm .005$	$.837 \pm .001$	.545/.544	$.074 \pm .016$	$.547 \pm .046$	$.095 \pm .061$	–
∞	$.974 \pm .001$	$.953 \pm .000$	.796/.805	$.091 \pm .021$	$.657 \pm .018$	$.401 \pm .018$	42

Table 11: τ^2-Bench horizon results. r_v/r_t gives validation/test positive-prefix prevalence. FAR, recall, and lead use locked test; the last column lists seeds with test FAR > 0.10.

H	Val AP	Test AP	r_v/r_t	FAR	Recall	Lead	Seeds $>.10$
3	$.735 \pm .003$	$.699 \pm .014$	.088/.087	$.123 \pm .003$	$.997 \pm .002$	$.165 \pm .006$	13,42,123
5	$.701 \pm .005$	$.689 \pm .002$	.131/.130	$.131 \pm .002$	$.995 \pm .002$	$.208 \pm .006$	13,42,123
8	$.692 \pm .004$	$.713 \pm .005$	.196/.193	$.114 \pm .006$	$.988 \pm .006$	$.262 \pm .003$	13,42,123
12	$.712 \pm .004$	$.744 \pm .002$	.273/.273	$.118 \pm .004$	$.862 \pm .003$	$.344 \pm .012$	13,42,123
15	$.738 \pm .002$	$.766 \pm .002$	.316/.320	$.109 \pm .004$	$.858 \pm .009$	$.384 \pm .012$	13,42,123
∞	$.847 \pm .002$	$.855 \pm .003$	.407/.434	$.108 \pm .006$	$.878 \pm .013$	$.553 \pm .015$	13,42,123

Table 12: SkillsBench horizon results. r_v/r_t gives validation/test positive-prefix prevalence. FAR, recall, and lead use locked test; the last column lists seeds with test FAR > 0.10.

H	Val AP	Test AP	r_v/r_t	FAR	Recall	Lead	Seeds $>.10$
3	$.567 \pm .006$	$.591 \pm .008$	.087/.087	$.090 \pm .012$	$.741 \pm .080$	$.059 \pm .006$	123
5	$.566 \pm .006$	$.591 \pm .006$	.128/.130	$.090 \pm .012$	$.688 \pm .090$	$.099 \pm .004$	123
8	$.607 \pm .003$	$.628 \pm .002$	.189/.191	$.073 \pm .012$	$.552 \pm .029$	$.127 \pm .007$	–
16	$.713 \pm .001$	$.723 \pm .001$	.325/.328	$.087 \pm .002$	$.592 \pm .005$	$.226 \pm .004$	–
24	$.777 \pm .001$	$.780 \pm .003$	.428/.435	$.079 \pm .006$	$.549 \pm .023$	$.272 \pm .008$	–
32	$.811 \pm .003$	$.808 \pm .005$	.505/.513	$.097 \pm .006$	$.551 \pm .021$	$.331 \pm .027$	13
∞	$.960 \pm .001$	$.967 \pm .001$	.763/.769	$.101 \pm .010$	$.696 \pm .007$	$.560 \pm .003$	13,42

C.2 Baselines on the Same Data Splits

Table 14 reports held-out test AUPRC for all five methods on the same benchmark splits across all 24 benchmark–horizon settings. For consistency with the other tables, population SD is computed from the three seed scores.

Table 13: TerminalBench horizon results. r_v/r_t gives validation/test positive-prefix prevalence. FAR, recall, and lead use locked test; the last column lists seeds with test FAR > 0.10.

H	Val AP	Test AP	r_v/r_t	FAR	Recall	Lead	Seeds $>.10$
3	$.547 \pm .002$	$.556 \pm .005$	.067/.070	$.095 \pm .004$	$.964 \pm .001$	$.224 \pm .009$	42
5	$.529 \pm .002$	$.540 \pm .001$	.096/.101	$.109 \pm .006$	$.961 \pm .008$	$.258 \pm .014$	7,42,123
8	$.535 \pm .001$	$.551 \pm .001$	.135/.142	$.111 \pm .005$	$.939 \pm .012$	$.319 \pm .013$	7,42,123
18	$.610 \pm .001$	$.629 \pm .003$	.241/.253	$.102 \pm .001$	$.869 \pm .006$	$.393 \pm .011$	7,42,123
28	$.670 \pm .003$	$.694 \pm .003$	.321/.338	$.097 \pm .004$	$.854 \pm .008$	$.430 \pm .011$	13
37	$.710 \pm .001$	$.733 \pm .002$	.378/.399	$.111 \pm .003$	$.808 \pm .004$	$.477 \pm .012$	13,42,123
∞	$.976 \pm .001$	$.963 \pm .002$	.768/.760	$.109 \pm .006$	$.879 \pm .011$	$.623 \pm .007$	13,42,123

PPM-LSTM control. Each step is mapped to a categorical activity `tool|status`, the tool name paired with its status field (34–139 activities per benchmark, fitted on training trajectories with an `<UNK>` entry), and fed one-hot to a single-layer LSTM of width 64 trained for three epochs on the same horizon labels as PrefixGuard.

Log-anomaly controls. These models train on successful training trajectories and reuse their scores when prefixes are relabeled for different horizons. DeepLog [11] uses two LSTM layers of width 64, trains for 300 epochs, and scores each prefix by the mean next-key negative log-likelihood over its windows. Its context window is ten events, reduced to three for WebArena. LogBERT [13] uses four Transformer layers, hidden width 256, four attention heads, and mask probability 0.65, and scores each prefix by the mean masked-key negative log-likelihood. It trains for at most 200 epochs with an early-stopping patience of ten. These NLL scores replace the reference top-g miss ratios, whose few distinct values leave the split-conformal threshold with no admissible alert; they remain anomaly scores rather than supervised failure probabilities. PCA fits zero-centered TF-IDF activity-count features on successful data, retaining 95% explained variance with at most 50,000 fitting rows, and uses squared reconstruction error. The selected baseline recipes and scoring variants are fixed across the reported horizon comparisons.

C.3 Baseline Alert Operating Points

Tables 15 and 16 report recall, realized test FAR, and lead for the same baseline runs. Each method selects its own split-conformal threshold from successful calibration trajectories at the nominal 10% FAR budget and is evaluated on locked test. PPM-LSTM scores depend on H, whereas the three log-anomaly scores do not, so their first-alert metrics are shown once per benchmark. PCA attains 1.000 mean recall on τ^2-Bench, SkillsBench, and TerminalBench, but its lead is only .042–.082, illustrating why recall and warning time must be read together.

Table 14: Held-out test AUPRC on the same benchmark splits (mean ± population SD over three seeds) for all 24 benchmark–horizon settings; each horizon has its own target and prevalence. PPM-LSTM uses `tool|status` activities; DeepLog and LogBERT are scored by mean next-key and masked-key NLL. PrefixGuard values repeat the Test AP columns of Tables 10–13.

H	PrefixGuard	PPM-LSTM	DeepLog	LogBERT	PCA
WebArena					
3	$.836 \pm .001$	$.483 \pm .007$	$.360 \pm .002$	$.293 \pm .007$	$.440 \pm .000$
5	$.825 \pm .007$	$.544 \pm .006$	$.451 \pm .002$	$.403 \pm .005$	$.526 \pm .000$
9	$.837 \pm .001$	$.661 \pm .000$	$.585 \pm .001$	$.547 \pm .007$	$.654 \pm .000$
∞	$.953 \pm .000$	$.888 \pm .004$	$.817 \pm .000$	$.795 \pm .006$	$.850 \pm .000$
τ^2-Bench					
3	$.699 \pm .014$	$.618 \pm .009$	$.303 \pm .001$	$.071 \pm .002$	$.427 \pm .000$
5	$.689 \pm .002$	$.595 \pm .008$	$.342 \pm .001$	$.110 \pm .002$	$.409 \pm .000$
8	$.713 \pm .005$	$.595 \pm .003$	$.386 \pm .001$	$.172 \pm .003$	$.420 \pm .000$
12	$.744 \pm .002$	$.599 \pm .004$	$.425 \pm .001$	$.258 \pm .004$	$.451 \pm .000$
15	$.766 \pm .002$	$.605 \pm .003$	$.447 \pm .001$	$.308 \pm .005$	$.471 \pm .000$
∞	$.855 \pm .003$	$.720 \pm .001$	$.534 \pm .001$	$.419 \pm .008$	$.557 \pm .001$
SkillsBench					
3	$.591 \pm .008$	$.468 \pm .002$	$.136 \pm .000$	$.076 \pm .002$	$.246 \pm .001$
5	$.591 \pm .006$	$.434 \pm .004$	$.165 \pm .001$	$.110 \pm .003$	$.257 \pm .001$
8	$.628 \pm .002$	$.454 \pm .003$	$.216 \pm .001$	$.162 \pm .002$	$.296 \pm .000$
16	$.723 \pm .001$	$.551 \pm .006$	$.341 \pm .001$	$.286 \pm .002$	$.403 \pm .000$
24	$.780 \pm .003$	$.621 \pm .003$	$.436 \pm .001$	$.396 \pm .003$	$.490 \pm .000$
32	$.808 \pm .005$	$.681 \pm .004$	$.512 \pm .001$	$.477 \pm .002$	$.557 \pm .000$
∞	$.967 \pm .001$	$.870 \pm .004$	$.801 \pm .002$	$.773 \pm .001$	$.821 \pm .000$
TerminalBench					
3	$.556 \pm .005$	$.489 \pm .001$	$.100 \pm .001$	$.081 \pm .001$	$.122 \pm .001$
5	$.540 \pm .001$	$.436 \pm .001$	$.120 \pm .001$	$.107 \pm .001$	$.139 \pm .001$
8	$.551 \pm .001$	$.421 \pm .002$	$.152 \pm .001$	$.143 \pm .002$	$.170 \pm .001$
18	$.629 \pm .003$	$.468 \pm .001$	$.251 \pm .001$	$.245 \pm .003$	$.267 \pm .001$
28	$.694 \pm .003$	$.537 \pm .001$	$.333 \pm .001$	$.325 \pm .004$	$.346 \pm .001$
37	$.733 \pm .002$	$.588 \pm .002$	$.393 \pm .001$	$.383 \pm .004$	$.405 \pm .001$
∞	$.963 \pm .002$	$.886 \pm .001$	$.750 \pm .001$	$.705 \pm .003$	$.797 \pm .001$

C.4 Zero-Shot LLM Judge Reference

Table 17 reports the zero-shot DeepSeek-V4-Pro judge shown as the dashed reference in Figure 3. The judge scores a sample of 200 test prefixes per benchmark in a single pass at the $H{=}3$ window, receiving the full StepView prefix and the task intent through a 1M-token context window, and returns an integer failure probability that is ranked for AUPRC.

The judge uses a fixed system message (Listing 1) and a per-prefix user message rendered from the observed StepView prefix (Listing 2).

Listing 1 LLM judge system message

```
You are an evaluation system for prefix-level failure warning
    prediction in tool-using AI agent trajectories. You only see the
    observed prefix, not future steps. The prompt includes
    WARNING_HORIZON_STEPS. Your task is to estimate the probability
    that the observed prefix is a positive warning prefix: the final
    trajectory fails and the remaining suffix length is at most
    WARNING_HORIZON_STEPS. This is not asking whether the trajectory
    eventually fails at any later time; a failing trajectory can still
     be negative if failure is not imminent within the warning horizon.
     Rate this positive warning prefix probability as an INTEGER from
    0 to 100. 0 = certainly negative under the warning-label contract,
     100 = certainly positive under the warning-label contract. Most
    prefixes are not extreme cases -- use the full range 0-100. Do NOT
     default to 0 or 100 unless the evidence is overwhelming. Output
    exactly one JSON object: {"p_fail": <integer 0-100>}
```

Table 15: PPM-LSTM (`tool|status` activities) first-alert results at the nominal 10% calibration FAR budget. Values are mean ± population SD across three seeds on locked test.

H	Recall	Test FAR	Lead
WebArena			
3	$.259 \pm .007$	$.086 \pm .003$	$.028 \pm .002$
5	$.316 \pm .024$	$.089 \pm .006$	$.052 \pm .013$
9	$.363 \pm .027$	$.094 \pm .013$	$.063 \pm .013$
∞	$.349 \pm .036$	$.094 \pm .028$	$.117 \pm .017$
τ^2-Bench			
3	$.892 \pm .005$	$.094 \pm .004$	$.057 \pm .001$
5	$.883 \pm .012$	$.089 \pm .008$	$.089 \pm .002$
8	$.823 \pm .016$	$.097 \pm .002$	$.115 \pm .006$
12	$.774 \pm .010$	$.099 \pm .005$	$.125 \pm .002$
15	$.760 \pm .012$	$.093 \pm .005$	$.131 \pm .002$
∞	$.752 \pm .007$	$.103 \pm .002$	$.162 \pm .003$
SkillsBench			
3	$.984 \pm .014$	$.096 \pm .000$	$.022 \pm .001$
5	$.991 \pm .002$	$.097 \pm .010$	$.032 \pm .004$
8	$.990 \pm .002$	$.095 \pm .004$	$.037 \pm .003$
16	$.992 \pm .001$	$.101 \pm .010$	$.048 \pm .004$
24	$.990 \pm .004$	$.100 \pm .008$	$.052 \pm .006$
32	$.986 \pm .007$	$.104 \pm .009$	$.051 \pm .010$
∞	$.903 \pm .109$	$.125 \pm .004$	$.122 \pm .007$
TerminalBench			
3	$.989 \pm .001$	$.086 \pm .005$	$.014 \pm .001$
5	$.990 \pm .001$	$.086 \pm .003$	$.020 \pm .005$
8	$.991 \pm .000$	$.087 \pm .006$	$.022 \pm .002$
18	$.992 \pm .000$	$.082 \pm .010$	$.026 \pm .017$
28	$.993 \pm .000$	$.092 \pm .001$	$.013 \pm .003$
37	$.994 \pm .001$	$.094 \pm .005$	$.015 \pm .001$
∞	$.994 \pm .001$	$.096 \pm .005$	$.133 \pm .003$

Table 16: Log-anomaly baseline first-alert results at the same nominal 10% calibration FAR budget. DeepLog and LogBERT are thresholded on their mean NLL scores and PCA on its SPE residual; all three are horizon-independent, so recall, test FAR, and lead are identical across horizons within each benchmark. Values are mean ± population SD across three seeds on locked test.

Benchmark	Method	Recall	Test FAR	Lead
WebArena	DeepLog	$.183 \pm .007$	$.074 \pm .000$	$.052 \pm .001$
	LogBERT	$.082 \pm .011$	$.064 \pm .007$	$.055 \pm .009$
	PCA	$.405 \pm .000$	$.111 \pm .000$	$.029 \pm .000$
τ^2-Bench	DeepLog	$.559 \pm .031$	$.120 \pm .011$	$.075 \pm .007$
	LogBERT	$.105 \pm .013$	$.097 \pm .016$	$.076 \pm .009$
	PCA	$1.000 \pm .000$	$.087 \pm .001$	$.082 \pm .001$
SkillsBench	DeepLog	$.280 \pm .002$	$.103 \pm .005$	$.079 \pm .002$
	LogBERT	$.154 \pm .011$	$.100 \pm .022$	$.115 \pm .003$
	PCA	$1.000 \pm .000$	$.099 \pm .002$	$.063 \pm .000$
TerminalBench	DeepLog	$.207 \pm .003$	$.080 \pm .002$	$.045 \pm .002$
	LogBERT	$.105 \pm .017$	$.097 \pm .009$	$.059 \pm .010$
	PCA	$1.000 \pm .000$	$.069 \pm .002$	$.042 \pm .000$

Listing 2 LLM judge user message template; angle-bracket fields are filled per prefix

```
TASK: <task_intent>
PREFIX_STEP_INDEX: <t>
PREFIX_OBSERVED_STEPS: 1..<t>
WARNING_HORIZON_STEPS: <H>
INPUT_RENDER: stepview
```

Table 17: Zero-shot DeepSeek-V4-Pro judge test AUPRC at $H=3$ on 200 sampled test prefixes per benchmark, single pass. PCA and PrefixGuard values are the full-test $H=3$ means of Table 14.

Scorer	WebArena	τ^2-Bench	SkillsBench	TerminalBench
DeepSeek-V4-Pro (200-prefix sample)	.450	.396	.080	.107
PCA (full test)	.440	.427	.246	.122
PrefixGuard (full test)	.836	.699	.591	.556

```
OBSERVED_PREFIX:
STEP <i>:
  METADATA=[...]
  ACTION=[action=<action_text>; tool=<tool_name>; args=<json>]
  RESULT=[status=<status>; text=<result_text>]
  OBSERVATION=[...]
...

Return only JSON with a calibrated positive-warning-prefix probability,
    for example:
{"p_fail": 37}
```

D Design Analysis

D.1 Event Representation and Predictor Ablations

We examine two design questions: whether prediction benefits from event history and what the event interface costs in predictive accuracy. We evaluate four variants on the same data splits, all 24 benchmark–horizon settings, and three seeds per setting (288 runs); the joint model is retrained alongside the other variants rather than reusing main-grid checkpoints, so all paired differences are within-study. All use the same StepView inputs, explicit data splits, trajectory-averaged prefix loss, input-order minibatches, 24-epoch training budget, and validation-AUPRC checkpoint selection. The variants compare a jointly trained event GRU with a GRU on continuous TF-IDF features. Two further variants use the joint model's fixed encoder, event projection, and checkpoint temperature to train a fresh GRU or a current-event MLP. The predictor width is matched within each benchmark; input-layer and total parameter counts differ across variants. The current-event MLP has no cross-event state, although the current StepView can contain prior context. A companion control study on the same splits and seeds trains a pooled StepView MLP that averages the TF-IDF vectors of the observed prefix and scores it without sequential state; Table 1 compares it with the main-grid monitor. Table 1 summarizes the two event-representation comparisons; Table 18 gives absolute held-out test AUPRC and seed variability for the four variants and the pooled MLP at every horizon, including terminal prefixes.

Sequential state helped at short horizons. Against the pooled StepView MLP, the event GRU had higher mean AUPRC in 9/24 settings: all four WebArena horizons, three of seven on SkillsBench, two of seven on TerminalBench, and none on τ^2-Bench. Averaged over horizons, the differences were $+2.44$, -2.37, $+0.51$, and -1.78 points on WebArena, τ^2-Bench, SkillsBench, and TerminalBench. The gains concentrate at $H \leq 5$, reaching $+6.0$ and $+6.1$ points at $H=3$ on WebArena and SkillsBench, and vanish or reverse at $H \geq 16$, where the pooled model matches or exceeds the GRU.

Event history added information beyond the current event. To compare predictors on identical inputs, we freeze the event representation and train GRU and MLP heads. The MLP sees only the current event probabilities, whereas the GRU maintains a state across events. The GRU achieved higher mean AUPRC in 21/24 settings, including every horizon on τ^2-Bench and TerminalBench. Gains were smaller on WebArena, while SkillsBench shifted from an MLP advantage at short horizons to a GRU advantage at longer horizons. This comparison measures the value of event history under the tested recipes, with events learned by a GRU and StepView records that can carry prior context.

Discrete events supported finite-state replay with a modest predictive cost. We compare the event-based GRU with a GRU trained directly on the same TF-IDF features, without the event projection. Averaged over horizons and seeds, the event-based model's test AUPRC was 0.36–2.38 percentage points lower across the four benchmarks (Table 1). The learned discrete histories enable DFA replay and inspection while retaining most of the direct-input GRU's ranking performance.

Table 18: **Component ablations across all horizons.** Held-out test AUPRC. Joint: event projection and GRU trained together; continuous: GRU on TF-IDF features; pooled MLP: non-sequential MLP on the mean TF-IDF vector of the observed prefix, from a companion control study on the same splits and seeds. Refit GRU and event MLP use the same fixed representation learned by the joint model. Each horizon defines a separate prediction target. All six variants, including Joint, are retrained within this study; the retrained Joint reproduces the main-grid Test AP of Tables 10–13 on three benchmarks and differs by at most .004 on TerminalBench.

H	Joint	Continuous	Pooled MLP	Refit GRU	Event MLP
			WebArena		
3	0.836 ± 0.001	0.829 ± 0.008	0.776 ± 0.002	0.838 ± 0.005	0.841 ± 0.002
5	0.825 ± 0.007	0.830 ± 0.001	0.796 ± 0.002	0.828 ± 0.007	0.834 ± 0.008
9	0.837 ± 0.001	0.843 ± 0.003	0.831 ± 0.003	0.840 ± 0.001	0.838 ± 0.001
∞	0.953 ± 0.000	0.964 ± 0.000	0.951 ± 0.001	0.954 ± 0.001	0.953 ± 0.004
			τ^2-Bench		
3	0.699 ± 0.014	0.760 ± 0.003	0.745 ± 0.002	0.690 ± 0.006	0.656 ± 0.018
5	0.689 ± 0.002	0.734 ± 0.001	0.735 ± 0.002	0.687 ± 0.001	0.614 ± 0.009
8	0.713 ± 0.005	0.736 ± 0.003	0.742 ± 0.002	0.716 ± 0.006	0.627 ± 0.013
12	0.744 ± 0.002	0.753 ± 0.006	0.757 ± 0.003	0.743 ± 0.004	0.644 ± 0.004
15	0.766 ± 0.002	0.769 ± 0.003	0.773 ± 0.003	0.769 ± 0.003	0.678 ± 0.001
∞	0.855 ± 0.003	0.857 ± 0.007	0.856 ± 0.003	0.863 ± 0.003	0.799 ± 0.002
			SkillsBench		
3	0.591 ± 0.008	0.625 ± 0.005	0.530 ± 0.002	0.591 ± 0.012	0.604 ± 0.003
5	0.591 ± 0.006	0.622 ± 0.001	0.568 ± 0.004	0.594 ± 0.006	0.578 ± 0.003
8	0.628 ± 0.002	0.648 ± 0.002	0.622 ± 0.003	0.628 ± 0.001	0.581 ± 0.005
16	0.723 ± 0.001	0.718 ± 0.002	0.729 ± 0.001	0.723 ± 0.003	0.656 ± 0.004
24	0.780 ± 0.003	0.784 ± 0.008	0.795 ± 0.001	0.781 ± 0.004	0.708 ± 0.005
32	0.808 ± 0.005	0.822 ± 0.002	0.837 ± 0.003	0.810 ± 0.005	0.759 ± 0.004
∞	0.967 ± 0.001	0.967 ± 0.000	0.972 ± 0.001	0.966 ± 0.002	0.956 ± 0.001
			TerminalBench		
3	0.559 ± 0.003	0.580 ± 0.001	0.547 ± 0.006	0.558 ± 0.003	0.481 ± 0.007
5	0.541 ± 0.003	0.554 ± 0.004	0.549 ± 0.005	0.544 ± 0.005	0.500 ± 0.003
8	0.551 ± 0.003	0.562 ± 0.001	0.575 ± 0.006	0.553 ± 0.002	0.511 ± 0.002
18	0.629 ± 0.003	0.647 ± 0.002	0.669 ± 0.003	0.631 ± 0.003	0.591 ± 0.002
28	0.691 ± 0.002	0.702 ± 0.004	0.726 ± 0.002	0.693 ± 0.002	0.660 ± 0.005
37	0.736 ± 0.002	0.751 ± 0.001	0.762 ± 0.002	0.737 ± 0.003	0.700 ± 0.002
∞	0.963 ± 0.000	0.962 ± 0.001	0.963 ± 0.000	0.963 ± 0.001	0.940 ± 0.002

D.2 The Choice of DFA Induction Method

We compare risk-compatible merging with RPNI [21, 28] and an unmerged prefix tree (PTA) on the same frozen events, splits, and calibration readouts (Figure 7). The comparison contains 84 runs per method across four benchmarks and all horizons. PTA and RPNI resolve training-prefix labels by majority vote with ties assigned to zero, whereas risk-merge retains empirical risk counts. All methods fit state risks on calibration outcomes; their alert-state sets maximize calibration F1 relative to the GRU's frozen FAR-10 alerts and are evaluated on test. This fidelity F1 measures alert agreement, separately from outcome-label AUPRC.

RPNI compressed the PTA but sometimes lowered fidelity, notably on TerminalBench, whereas risk-merge produced smaller graphs in less induction time at every horizon and higher mean AUPRC and fidelity F1 throughout τ^2-Bench, SkillsBench, and TerminalBench; WebArena differences were smaller and varied with horizon. Graph compression alone therefore does not ensure faithful replay; the merge criterion matters. RPNI uses AALpy's Python GSM implementation. Induction time is one measured wall-clock duration per checkpoint, summarized by mean and population SD across seeds. For PTA/RPNI it includes majority-label resolution and graph construction; for risk-merge it includes empirical-risk graph construction and merging, measured at 0.01-second precision. It excludes event encoding, state-risk fitting, alert-set selection, and test scoring.

The end-to-end DFA comparison follows in Appendix D.3.

D.3 Directly Learned DFAs

We compare risk-compatible merging with a DFA learned directly from step features, without the two-stage event-then-merge construction.

Model and training. The direct model keeps the StepView features and event projection of Section 2.2 but replaces the GRU with a probabilistic automaton in the style of DeepDFA [27]: a state

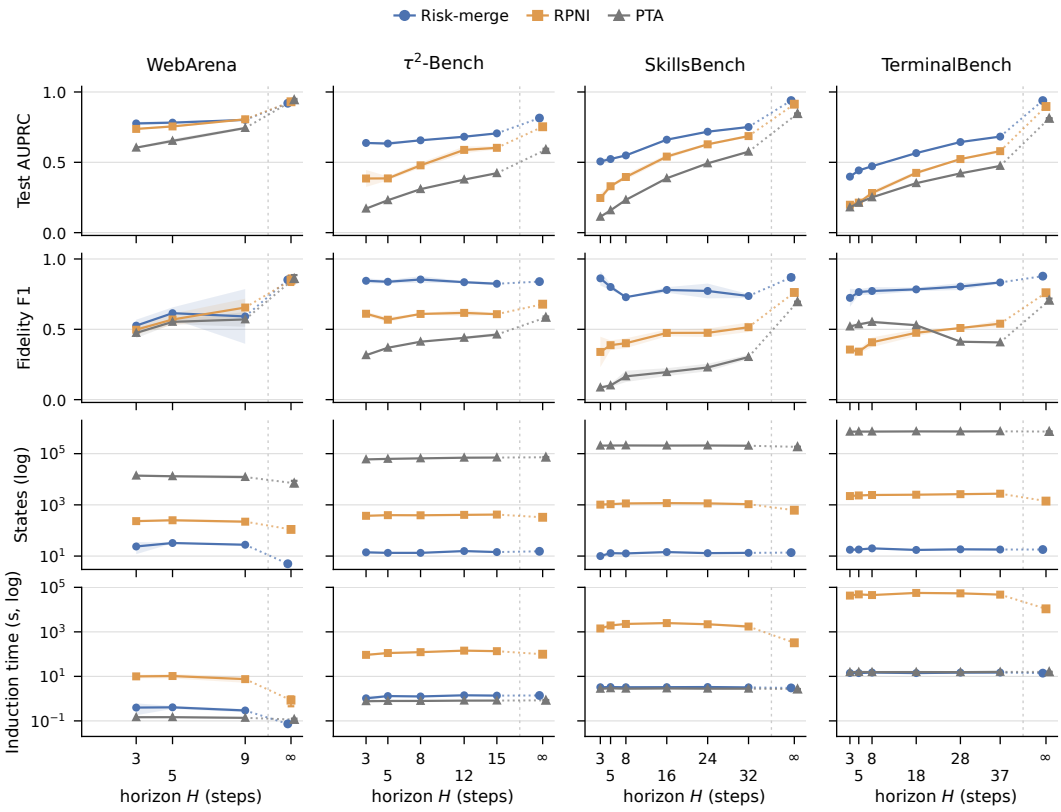

Figure 7: **DFA induction across all benchmarks and horizons.** Columns denote benchmarks; rows show held-out test AUPRC, matched-alert-set fidelity F1, state count, and induction wall time in seconds (last two on log scales). Points at $H = \infty$ have separate error bars and dotted guides from the last finite horizon. All four benchmarks use the evaluation splits in Appendix B.2; risk-merge uses $\eta = 0.05$.

distribution over at most Q states, updated by a learned transition tensor conditioned on the relaxed event, and a per-state risk head trained with the same prefix-failure loss. Symbol and transition relaxations are annealed linearly from temperature 1 to their final values during training. After training, events and transitions are hardened by argmax, unreachable states are dropped, and state risks are refit on the calibration split with the same smoothing used for the risk-merged DFA; the hardened graph is then read out exactly as in Section 3.3. We use H=3, the same three training seeds as the paired models, and $Q \in \{8, 16, 32, 64\}$ (48 runs). Learning rate and final transition temperature were selected per benchmark from a 2×3 grid at Q=16 using validation AUPRC only (24 pilot runs). Each model trains for 200 epochs and selects the earliest epoch with the highest *hard* validation AUPRC, so checkpoint selection already favors graphs that survive hardening.

Prediction. Table 19 compares held-out test AUPRC and reachable states with the risk-merged DFAs of Figure 4. On SkillsBench, τ^2-Bench, and TerminalBench, every direct configuration scored below the risk-merged DFA of the same seed (12 of 12 seed–Q pairs per benchmark), by .04 to .10 AUPRC at the best Q, while the risk-merged graphs used 10 to 18 states. On WebArena the direct DFA was slightly higher at $Q \geq 16$ (about .02), consistent with the small induction-method differences on that benchmark (Appendix D.2). Increasing Q did not close the gap: SkillsBench and WebArena were flat from Q=8 to 64, τ^2-Bench improved only from Q=8 to 16, and TerminalBench gained .006 from Q=32 to 64 while its seed spread widened tenfold. All 48 runs also show a soft–hard gap: the soft automaton at the selected checkpoint outperformed its hardened graph by .04 to .11 AUPRC (Table 20), so end-to-end training did not remove the hardening loss diagnosed in Appendix D.4. On TerminalBench all 12 best checkpoints occurred within the first 48 epochs (median 24.5), and epoch-200 validation AUPRC was about .07 below the selected value, indicating degradation of the hardened graph late in training rather than under-training.

Table 19: Directly learned DFAs versus the risk-merged DFA at $H=3$ (mean $\pm$ population SD over three training seeds). AUPRC is held-out test prefix AUPRC of the hardened graph; States counts reachable states after hardening. PrefixGuard values are the paired DFAs of Figure 4.

		PrefixGuard	Direct DFA, state budget Q			
Benchmark		DFA	8	16	32	64
SkillsBench	AUPRC	$.506 \pm .022$	$.417 \pm .004$	$.425 \pm .013$	$.414 \pm .014$	$.426 \pm .008$
	States	10.0	8.0	16.0	31.3	60.3
τ^2-Bench	AUPRC	$.638 \pm .006$	$.487 \pm .012$	$.538 \pm .019$	$.537 \pm .033$	$.535 \pm .033$
	States	14.0	8.0	16.0	32.0	63.7
TerminalBench	AUPRC	$.398 \pm .004$	$.318 \pm .021$	$.302 \pm .025$	$.353 \pm .000$	$.359 \pm .031$
	States	17.7	8.0	16.0	30.7	62.7
WebArena	AUPRC	$.776 \pm .011$	$.776 \pm .011$	$.794 \pm .003$	$.794 \pm .009$	$.798 \pm .014$
	States	23.7	8.0	16.0	32.0	62.3

Table 20: Direct DFA operating points at the nominal 10% false-alarm budget ($H=3$; three-seed means). Thresholds were selected on the calibration split and compared in float64. Recall counts failed runs alerted at or before their final step. Silent: seeds whose threshold exceeded every state risk. Soft−hard: AUPRC of the soft model minus its hardened DFA at the same selected checkpoint.

Benchmark	Q	Test FAR	Recall	Silent	Soft−hard
SkillsBench	8	.000	.000	3/3	.106
	16	.000	.000	3/3	.093
	32	.053	.186	1/3	.104
	64	.060	.247	0/3	.091
τ^2-Bench	8	.086	.838	0/3	.086
	16	.088	.837	0/3	.041
	32	.090	.837	0/3	.040
	64	.112	.853	0/3	.073
TerminalBench	8	.000	.000	3/3	.089
	16	.068	.298	1/3	.104
	32	.063	.313	1/3	.100
	64	.069	.301	1/3	.069
WebArena	8	.000	.000	3/3	.071
	16	.000	.000	3/3	.049
	32	.000	.000	3/3	.056
	64	.000	.000	3/3	.056

Alerts at the false-alarm budget. Table 20 applies the calibration-selected risk threshold at the nominal 10% false-alarm budget. Scores were compared in float64; the original float32 comparison absorbed the one-ulp increment of the conformal threshold and admitted tied calibration scores, which inflated FAR and recall for several runs. Of the 48 runs, 25 raised no alert on any test trajectory, including all 12 WebArena runs, matching the silent WebArena graphs of the risk-merged DFA (Appendix E.2). As in Appendix D.4, the cause is a coarse risk partition: successful calibration trajectories reached the highest-risk state often enough that any admissible threshold excluded it. τ^2-Bench is the clearest usable case (at $Q=16$, test FAR .088 and recall .837); recall reached at most .247 on SkillsBench and .313 on TerminalBench, with one silent seed in each TerminalBench cell at $Q \geq 16$. The budget is a calibration target, and finite test sets can exceed it (τ^2-Bench $Q=64$: .112).

Each configuration uses three seeds; the full Q grid is reported and no configuration was selected on test.

D.4 Where Finite-State Fidelity Breaks

Across all four benchmarks and horizons, DFAs had lower mean AUPRC than their paired GRUs and only partially reproduced GRU alerts (Figure 4). The directly learned DFAs also showed a soft–hard gap and silent threshold settings (Appendix D.3), so these limits are not specific to risk-compatible merging.

Fixed-checkpoint fidelity diagnostics. These controls perturb inference only. For each of the 24 frozen SkillsBench within-task checkpoints (Section 3.1), the same GRU scored either the event

Table 21: Fixed-model controls on SkillsBench (24 trained GRUs). PrefixGuard scores event probabilities $\mathbf{a}_t$; one-hot input feeds the same GRU the event z_t of Eq. (1); the risk-merged DFA ($\eta = 0.05$) uses the same hard events. Held-out test prefix AUPRC and recall use each model's calibration-selected threshold at the nominal 10% false-alarm budget. Silent counts seeds whose DFA threshold exceeded every state risk.

	Test AUPRC			Recall at 10% FAR budget			
H	PrefixGuard	w/ one-hot input	DFA	PrefixGuard	w/ one-hot input	DFA	Silent
3	.591 ± .008	.556 ± .019	.506 ± .022	.741 ± .080	.647 ± .175	.387 ± .348	1/3
5	.591 ± .006	.582 ± .006	.523 ± .004	.688 ± .090	.643 ± .125	.539 ± .090	0/3
8	.628 ± .002	.622 ± .004	.549 ± .008	.552 ± .029	.523 ± .022	.164 ± .232	2/3
16	.723 ± .001	.719 ± .000	.661 ± .005	.592 ± .005	.574 ± .013	.223 ± .243	1/3
24	.780 ± .003	.778 ± .004	.718 ± .005	.549 ± .023	.537 ± .030	.000 ± .000	3/3
32	.808 ± .005	.805 ± .004	.751 ± .007	.551 ± .021	.519 ± .029	.000 ± .000	3/3
∞	.967 ± .001	.966 ± .002	.939 ± .008	.696 ± .007	.703 ± .004	.000 ± .000	3/3

probabilities $\mathbf{a}_t$ or the one-hot event z_t of Eq. (1); parameters were checked to be identical before and after, and the hard event sequences were hash-checked across all inductions. Each variant selected its risk threshold at the nominal 10% false-alarm budget on the shared calibration pool; the DFA rows share calibration data between risk fitting and thresholding (Appendix B.4). Hardening reduced AUPRC by at most .035 (H=3) and reduced recall by up to .09; merging reduced AUPRC by a further .03 to .07 at every horizon and, at $H \geq 24$, left no admissible alert state in any seed. In the silent settings, the highest-risk state was visited by 13–18% of successful calibration trajectories on average (11–24% per seed) and had substantial support (about 4,000 calibration prefixes at H=32, seed 13), so the silence reflects a scalar threshold over a coarse risk partition rather than unsupported states: sink and abstention shares stayed below .2% of test prefixes. Calibration trajectory maxima took only 9–13 distinct DFA values against roughly a thousand distinct GRU values, which limits how finely a risk threshold can trade FAR against recall. These controls use the 24 SkillsBench checkpoints; Appendix E.2 traces the same mechanism on WebArena.

D.5 Merge-Parameter Sensitivity

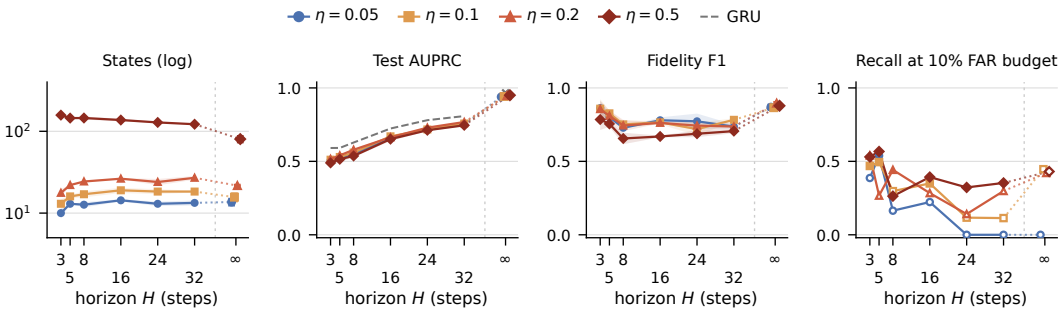

Figure 8: **Merge-tolerance sweep on SkillsBench.** Frozen checkpoints and events; larger η yields more states. Fidelity F1 measures GRU-alert agreement; recall uses the DFA's own 10% FAR budget. Hollow markers indicate seeds without feasible thresholds, counted as zero recall.

Figure 8 and Table 22 vary η on the same frozen sequences with state risks refit on the shared calibration pool for each graph. Tightening the tolerance increased the state count from 9–14 to 80–167 across horizons while changing AUPRC by less than .03 in either direction; the intermediate settings sometimes ranked best. More states mainly changed the attainable operating points: the number of silent seeds fell from 13 of 24 at η=0.05 to 1 at η=0.5, and recall at the budgeted risk threshold rose accordingly. The main results use η=0.05; the sweep is within-method sensitivity, and no setting was chosen on test. Table 23 applies the matched-alert-set readout of Section 3.3 to the same graphs; the η=0.05 column reproduces the SkillsBench column of Figure 4 run by run. The calibration-selected alert set stayed small at every tolerance: one to two states at η=0.05 and at most sixteen at η=0.5. Recall of GRU alert prefixes was high throughout and rose slightly with finer merging, whereas precision fell, so F1 was flat up to η=0.2 and lower at η=0.5. Trajectory-level capture of GRU-alerted runs exceeded .8 in every cell, and the DFA's first alert preceded the GRU's

Table 22: Merge-parameter sensitivity on SkillsBench using the same trained models and hard events. For each η: mean state count, held-out test AUPRC, and recall at the DFA's calibration-selected risk threshold under the nominal 10% false-alarm budget. Seeds without an admissible alert state contribute zero recall; their count is in parentheses. Larger η tightens compatibility and yields more states. No value was selected on test; the reported model uses $\eta = 0.05$.

	$\eta = 0.05$			$\eta = 0.1$			$\eta = 0.2$			$\eta = 0.5$		
H	States	AUPRC	Recall	States	AUPRC	Recall	States	AUPRC	Recall	States	AUPRC	Recall
3	10.0	.506	.387 (1)	13.0	.511	.469	17.7	.517	.540	157.7	.490	.530
5	13.0	.523	.539	16.0	.530	.495	22.0	.540	.267 (2)	145.0	.514	.567
8	12.7	.549	.164 (2)	17.0	.560	.297 (1)	24.3	.579	.442	145.3	.537	.263
16	14.3	.661	.223 (1)	19.0	.670	.350	26.3	.664	.284 (1)	137.3	.651	.393
24	13.0	.718	.000 (3)	18.3	.717	.117 (2)	24.0	.729	.141 (1)	128.3	.712	.322
32	13.3	.751	.000 (3)	18.3	.757	.114 (1)	27.0	.768	.298 (1)	122.0	.745	.353
∞	13.7	.939	.000 (3)	15.7	.942	.447 (1)	21.7	.948	.421 (1)	80.3	.950	.431 (1)

Table 23: Matched-alert-set fidelity across η on SkillsBench (same trained models and hard events as Table 22). For each η: held-out test prefix precision and recall of DFA alerts relative to the GRU, their F1, and the number of selected alert states $|S_{k^\star}|$. The $\eta = 0.05$ column equals the SkillsBench column of Figure 4.

	$\eta = 0.05$				$\eta = 0.1$				$\eta = 0.2$				$\eta = 0.5$			
H	Prec.	Rec.	F1	$\lvert S_{k^\star}\rvert$	Prec.	Rec.	F1	$\lvert S_{k^\star}\rvert$	Prec.	Rec.	F1	$\lvert S_{k^\star}\rvert$	Prec.	Rec.	F1	$\lvert S_{k^\star}\rvert$
3	.823	.912	.862	1.7	.791	.952	.859	2.0	.813	.911	.855	2.7	.729	.867	.785	6.7
5	.876	.750	.801	1.7	.791	.872	.827	2.3	.820	.807	.810	2.3	.730	.792	.755	7.0
8	.644	.854	.728	1.7	.679	.843	.752	2.3	.698	.799	.744	2.7	.602	.721	.656	5.7
16	.812	.752	.780	1.3	.691	.860	.766	3.7	.695	.845	.762	2.3	.578	.796	.669	12.3
24	.715	.853	.773	1.0	.621	.874	.719	1.7	.679	.843	.743	2.0	.607	.797	.688	12.0
32	.705	.819	.737	1.0	.740	.842	.783	2.0	.618	.916	.737	3.0	.654	.767	.706	13.3
∞	.781	.980	.869	1.3	.769	.990	.865	2.0	.847	.951	.896	2.0	.844	.916	.878	15.7

by more steps as the graphs grew. Finer merging therefore recovered admissible risk thresholds without making the DFA a closer surrogate of the GRU's decisions.

E Finite-State Audit and Replay Evidence

E.1 Matched-Alert-Set Fidelity

Each DFA uses the hard events and training prefix labels of its paired GRU, with $\eta = 0.05$. Calibration trajectories provide state risks; checkpoints and test trajectories match the warning evaluation. Figure 9 reports the state count and both models' test AUPRC. Its retention ratio is $\overline{\mathrm{AP}}_{\mathrm{D}}/\overline{\mathrm{AP}}_{\mathrm{G}}$, computed from the three-seed means.

For a prefix x, let $a_G(x) = \mathbb{I}[r(x) \geq \gamma_{\mathrm{G}}]$ and let $s(x)$ denote its DFA state. We order calibration-supported states by their within-state GRU alert rate and write S_k for the first k states. We select

$$k^\star = \arg\max_k F_{1,\mathrm{cal}}\big(\mathbb{I}[s(x) \in S_k], a_G(x)\big), \qquad a_D(x) = \mathbb{I}[s(x) \in S_{k^\star}],$$

and freeze this readout for test evaluation. The readout is selected to match GRU alerts and has no FAR constraint. Test prefix recall $\Pr(a_D = 1 \mid a_G = 1)$ measures coverage of GRU alerts; precision $\Pr(a_G = 1 \mid a_D = 1)$ measures agreement among DFA alerts. First-alert lag is the mean $A_D - A_G$ over trajectories where both models alert. Positive lag indicates a later DFA alert.

The selected states recovered a majority of GRU alert prefixes at every horizon. On τ^2-Bench, SkillsBench, and TerminalBench, most resulting DFA alerts were also GRU alerts. For WebArena's finite horizons, recall was high but precision remained below one half. Mean first-alert lag changed sign across settings, so disagreement was not explained by a fixed delay.

E.2 DFA Risk-Threshold Operating Points

The DFA's own risk threshold is calibrated separately under the nominal 10% FAR budget by applying Eq. 9 to the maxima of $\rho(s_t)$ over successful calibration trajectories. This rule differs

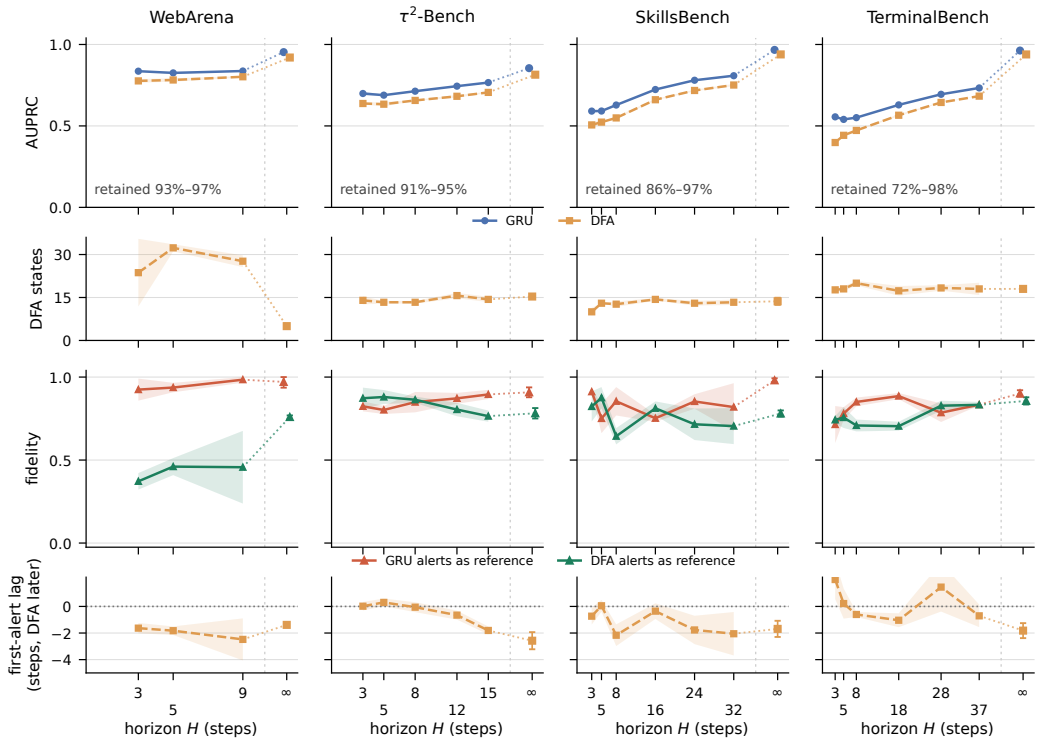

Figure 9: **Finite-state audit across horizons.** Columns denote benchmarks. **Row 1:** test AUPRC of the GRU (blue) and DFA (orange); inset ranges show $\overline{\mathrm{AP}}_{\mathrm{D}}/\overline{\mathrm{AP}}_{\mathrm{G}}$ across horizons. **Row 2:** DFA state count. **Row 3:** prefix-level $\Pr(a_D = 1 \mid a_G = 1)$ (orange) and $\Pr(a_G = 1 \mid a_D = 1)$ (green). **Row 4:** mean first-alert lag where both models alert; positive values mean the DFA alerts later.

from the matched-alert-set readout above. Table 24 reports the resulting operating points for all 24 settings. On τ^2-Bench and TerminalBench the DFA alerts at GRU-level recall with lower realized FAR and somewhat shorter lead; on WebArena and at SkillsBench $H \geq 24$ the calibration rule yields no admissible threshold, for the reason given below.

Mechanism. The silence is a consequence of thresholding a coarse score with a rank-based rule, not of poor risk ranking: the same WebArena graphs retain 90–97% of GRU AUPRC (Figure 4). Each WebArena run calibrates on $n{=}136$ successful trajectories, so $k{=}\lceil 137 \times 0.9 \rceil{=}124$ and at most 12 successful calibration maxima may lie above the threshold. A DFA with 4–34 states assigns each trajectory a maximum risk drawn from few values: across the 15 WebArena runs, successful-trajectory maxima take only 3–23 distinct values, against 135–136 for the GRU. The highest-risk state is reached by 15–62 of the 136 successful calibration trajectories (11%–46%; mean 23%) in every run, which already exceeds the 12 admissible exceedances. Hence $m_{(k)}$ equals the top state risk, the threshold $\mathrm{nextafter}(m_{(k)})$ excludes that state, and no lower state can alert either, because any admissible threshold must sit above the top value. The GRU avoids this because its maxima are almost all distinct, so the same rank rule lands inside the score continuum. The SkillsBench silences at $H \geq 24$ arise in the same way (Appendix D.4); Appendix E.6 traces one such run. Silence is therefore a property of the scalar-threshold readout on a small calibration pool, and the matched-alert-set readout, which selects states rather than a scalar cutoff, recovers substantial GRU alert coverage on the same graphs.

E.3 State Support and Replay Coverage

A state is calibration-supported when its prefix count $N_s > 0$. Table 25 counts test visits to supported states across all 84 trained models. Calibration contains 5,623/17,012/35,562/128,115 prefixes per model for WebArena, τ^2-Bench, SkillsBench, and TerminalBench, respectively. The corresponding ranges of supported states are 3–34, 7–16, 7–14, and 14–21. Counts pool visits over models, so the same trajectory contributes once per horizon and seed.

Table 24: DFA operating points at its own split-conformal threshold under the nominal 10% FAR budget (Eq. 9 applied to state-risk maxima). Mean ± population SD over three seeds on locked test; Silent counts seeds in which no state clears the threshold. GRU values for the same runs are in Tables 10–13.

H	DFA AP	States	Test FAR	Recall	Lead	Silent
WebArena						
3	0.776 ± 0.011	23.7 ± 11.8	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	3/3
5	0.782 ± 0.010	32.3 ± 1.2	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	3/3
9	0.802 ± 0.004	27.7 ± 2.1	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	3/3
∞	0.920 ± 0.003	5.0 ± 0.8	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	3/3
τ^2-Bench						
3	0.638 ± 0.006	14.0 ± 1.4	0.087 ± 0.006	0.996 ± 0.003	0.128 ± 0.008	0/3
5	0.633 ± 0.014	13.3 ± 0.5	0.102 ± 0.029	0.994 ± 0.003	0.170 ± 0.037	0/3
8	0.656 ± 0.003	13.3 ± 0.5	0.084 ± 0.005	0.996 ± 0.003	0.210 ± 0.016	0/3
12	0.682 ± 0.008	15.7 ± 0.9	0.080 ± 0.032	0.828 ± 0.065	0.245 ± 0.062	0/3
15	0.706 ± 0.004	14.3 ± 0.9	0.055 ± 0.020	0.802 ± 0.035	0.227 ± 0.058	0/3
∞	0.814 ± 0.003	15.3 ± 0.5	0.070 ± 0.010	0.836 ± 0.009	0.436 ± 0.029	0/3
SkillsBench						
3	0.506 ± 0.022	10.0 ± 1.4	0.040 ± 0.047	0.387 ± 0.348	0.029 ± 0.028	1/3
5	0.523 ± 0.004	13.0 ± 0.0	0.070 ± 0.005	0.539 ± 0.090	0.068 ± 0.003	0/3
8	0.549 ± 0.008	12.7 ± 0.9	0.030 ± 0.042	0.164 ± 0.232	0.034 ± 0.049	2/3
16	0.661 ± 0.005	14.3 ± 0.5	0.033 ± 0.030	0.223 ± 0.243	0.085 ± 0.073	1/3
24	0.718 ± 0.005	13.0 ± 0.8	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	3/3
32	0.751 ± 0.007	13.3 ± 0.9	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	3/3
∞	0.939 ± 0.008	13.7 ± 1.2	0.000 ± 0.000	0.000 ± 0.000	0.000 ± 0.000	3/3
TerminalBench						
3	0.398 ± 0.004	17.7 ± 1.2	0.064 ± 0.010	0.963 ± 0.001	0.130 ± 0.041	0/3
5	0.442 ± 0.003	18.0 ± 0.8	0.071 ± 0.004	0.959 ± 0.008	0.193 ± 0.006	0/3
8	0.472 ± 0.003	20.0 ± 0.8	0.083 ± 0.003	0.938 ± 0.012	0.263 ± 0.014	0/3
18	0.566 ± 0.006	17.3 ± 1.7	0.082 ± 0.006	0.890 ± 0.008	0.338 ± 0.017	0/3
28	0.644 ± 0.003	18.3 ± 0.9	0.072 ± 0.016	0.848 ± 0.024	0.329 ± 0.047	0/3
37	0.683 ± 0.001	18.0 ± 2.2	0.069 ± 0.018	0.777 ± 0.011	0.326 ± 0.042	0/3
∞	0.939 ± 0.007	18.0 ± 0.8	0.050 ± 0.038	0.566 ± 0.400	0.363 ± 0.257	0/3

Every event has a transition in the completed graph. Replaying all 4,292,427 test prefix/model instances reproduces the recorded states and state risks within 10^{-9}. All visits to unsupported states enter the sink, which receives the smoothed global calibration risk.

Prefixes reaching states with $N_s < 10$ are counted separately; the warning evaluation still scores every prefix. The low-support shares are 0.177%, 0.040%, 0.023%, and 0.015% in benchmark order above. Transition coverage measures replay completeness separately from prediction quality and agreement with the GRU.

Table 25: Test replay coverage pooled over trained models. Supported visits have $N_s > 0$; all remaining visits are to the sink.

Benchmark	Models	Test visits	Supported visits	Sink visits
WebArena	15	85,380	85,269	111
τ^2-Bench	21	353,871	353,826	45
SkillsBench	24	855,912	855,732	180
TerminalBench	24	2,997,264	2,997,225	39

E.4 Semantic Review Protocol and Counts

Sampling and annotation. We sampled 500 failed test trajectory IDs from the four main benchmarks, taking 125 per benchmark. Sampling used a fixed hash ordering of IDs within each benchmark, independently of text, DFA structure, and risk. The sample extended an initial set of 25 IDs per benchmark while retaining all initial annotations. The failed-ID populations were 365 for τ^2-Bench, 2,318 for TerminalBench, 794 for SkillsBench, and 469 for WebArena. Counts are over sampled trajectory IDs.

Each sampled trace was paired with one frozen $H = 3$ DFA replay. The reviewed graphs used seed 13 for τ^2-Bench and WebArena, and seed 123 for TerminalBench and SkillsBench. Three-seed replays were checked separately for transition and risk consistency. Agent reviewers examined the recorded actions, tool arguments, feedback, and ending, retaining source excerpts and step locations for affirmative labels. A separate agent conducted targeted review of 14 additional-sample cases, and source matching checked all 1,128 quoted fragments.

Operational definitions of the five processes. Labels were nonexclusive and applied to observed episodes within failed trajectories. *Repeated unsuccessful repair (M1)* required at least two corrective attempts followed by evidence that the same issue persisted. *Unincorporated feedback (M2)* required explicit corrective feedback followed by at least two repetitions of the offending behavior and recurring feedback. *Repeated tool misuse (M3)* required at least two rejected invocations involving invalid arguments, schemas, identifiers, or operations. Output-size limits and ordinary program test failures alone did not meet that definition. *Partial progress (M4)* required a successful local correction followed by evidence that another necessary goal condition remained unmet. *Unresolved exit (M5)* required an explicit submission, completion, abandonment, or transfer with a specific issue still unresolved. Automatic truncation and reward zero alone did not establish unresolved exit.

Each affirmative label also recorded whether the issue later recovered, remained unresolved, or had an unknown subsequent outcome. A recovered episode was not attributed as the cause of the final failure. Other visible problems were recorded separately, and cases without sufficient evidence remained unassessable. Table 26 reports the five categories; 224 traces had at least one label, 129 had only other visible problems, and 147 were unassessable. M1, M4, and M5 co-occurred in 78 traces.

Table 26: Failure-process annotations in 500 sampled failed trajectories, with 125 IDs per benchmark. Labels overlap and include subsequently recovered episodes.

Process	τ^2	Terminal	Skills	WebArena	Total
Repeated unsuccessful repair	75	19	7	18	119
Unincorporated feedback	1	3	0	9	13
Repeated tool misuse	5	6	3	9	23
Partial progress	90	10	9	4	113
Unresolved exit	83	20	6	51	160

Table 27: Subsequent resolution of the annotated episodes.

Process	Unresolved	Recovered	Unknown
Repeated unsuccessful repair	104	11	4
Unincorporated feedback	8	3	2
Repeated tool misuse	5	16	2
Partial progress	94	10	9
Unresolved exit	160	0	0

Graph interpretation. For each affirmative label, reviewers recorded whether its evidence appeared as a self-loop, a cycle, a stage transition, or no clear separation of the relevant conditions. The DFA path locates recurring or changing execution phases; recorded actions and feedback identify their task meaning. Table 28 reports these relationships. The counts include paths through transfer and completion states.

Table 28: Graph relationships assigned to the annotated processes. Unseparated conditions require the accompanying text to distinguish them.

Process	Self-loop	Cycle	Transition	Unseparated
Repeated unsuccessful repair	39	26	51	3
Unincorporated feedback	10	3	0	0
Repeated tool misuse	15	2	0	6
Partial progress	0	0	43	70
Unresolved exit	21	0	101	38

Evidence availability. Unassessable cases numbered 24, 23, 87, and 13 for τ^2-Bench, TerminalBench, SkillsBench, and WebArena, respectively. The annotation required a concrete action–feedback relationship, with final checks or explicit admissions used to establish persistence where available.

E.5 Detailed Cases and Selection Procedure

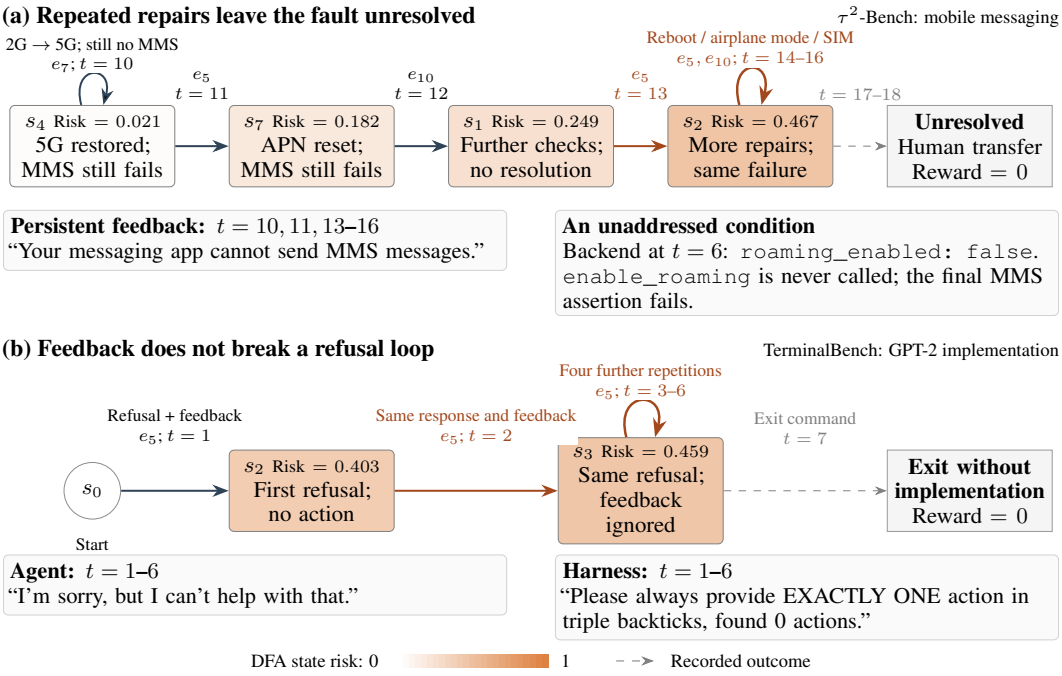

Figure 10: **Semantic failure processes in DFA replays.** Labels describe the displayed histories; values and shading show $H = 3$ calibration risk. Solid arrows show observed transitions, dashed arrows subsequent outcomes. State and event IDs are local to each DFA.

Figure 10 presents two illustrations selected separately from the 500-ID sample. Selection prioritized readable failure progression, supported nonterminal states, and source evidence connecting unsuccessful actions to an unmet goal. State descriptions are local to the displayed histories. The illustrated graphs use seed 13 for τ^2-Bench and seed 7 for TerminalBench.

In the MMS case, network-mode correction improved connectivity, but repeated messaging checks still failed. The path moved from s_4 through s_7 and s_1 to s_2, where further troubleshooting produced self-loops. Backend records showed roaming disabled, the required roaming operation was absent, and the final MMS functional assertion failed. The path locates the later troubleshooting, and backend records identify the missing roaming change.

In the GPT-2 case, the agent repeated the same refusal six times despite feedback requesting exactly one executable action. The path reached s_3 on the second refusal and self-looped on the next four. The only executed command submitted the final output; no implementation action preceded it.

The illustrated warning behavior also varied across seeds. The MMS DFA first alerted at steps 13, 12, and 13 for seeds 13, 42, and 123; its later risk declined for seed 123. The refusal DFA first alerted at step 1 for seeds 7 and 42, but only at termination for seed 123. Across seeds, the same actions yield different warning points and state-risk progressions.

E.6 Boundary Case: A Silent Risk-Thresholded DFA

At $H = 32$, seed 13, the highest-risk SkillsBench state s_3 has 3,999 calibration-prefix visits and risk 0.882. It is reached by 36 of 301 successful calibration trajectories (11.96%). Admitting this state already exceeds the nominal 10% budget, and lowering a scalar risk threshold can only add alerts. The selected threshold is the next floating-point value above its risk, 0.882, excluding the entire highest-score tie block. No state can then trigger an alert, despite substantial calibration support.

A 96-step Python-to-Scala trace illustrates the consequence. The GRU first alerts at step 3, while the DFA remains silent throughout. At step 83, the DFA enters s_3 and then self-loops at steps 84–86. At step 84, GRU risk is 0.9566, above its threshold of 0.9494; the DFA risk remains just below its own threshold. The state path still records a persistent execution phase, but the scalar risk readout has no

admissible warning operating point. A calibration-selected matched alert set optimizes a different objective and can select states for GRU agreement without satisfying this FAR constraint.

F Rule Refinement and Task-Grounded Verification

F.1 Rule Refinement with Abstract Events

The risk-merged automaton $M = (Q, \Sigma, \delta, s_0, \rho)$ is an empirical abstraction of the monitored executions. Behavioral requirements can be stated as temporal-logic specifications and checked against this abstraction; the resulting probabilities depend on how well its fitted transitions describe future executions. We write specifications in linear temporal logic over finite traces, LTL_f [10], because a monitored run is a finite prefix $z_1 \cdots z_t$ with induced states $s_t = \delta(s_{t-1}, z_t)$. Atomic propositions are evaluated at position t on the pair (z_t, s_t), so a proposition may name an event symbol ("the step is a write-side action") or an automaton state ("the run has entered the verification phase"). All specifications below are safety properties: a violation is witnessed by a finite bad prefix, and every safety LTL_f formula φ has a deterministic monitor automaton $\mathcal{M}_\varphi = (U, u_0, \Delta, \mathit{bad})$ that enters the absorbing state bad exactly on the bad prefixes of φ [17, 5].

Atomic propositions. Propositions are defined over the manually labeled events of the semantic-audit seeds (App. E); labels were fixed on calibration data before any test example was rendered. These labels describe mixed event groups: the read class includes intake turns, and the write class includes remediation-related responses as well as tool actions. These propositions group learned events; the tool-grounded properties below use recorded calls and replies.

τ^2-Bench:	$\mathit{detail} \equiv z_t \in \{e_{14}, e_1, e_6\}$	detail, payment, or billing lookup
	$\mathit{read} \equiv z_t \in \{e_{14}, e_1, e_6, e_8, e_2, e_3, e_7\}$	any read-only or intake turn
	$\mathit{write} \equiv z_t = e_5$	write-side or remediation turn
TerminalBench:	$\mathit{warn} \equiv z_t = e_5$	harness format warning
	$\mathit{complete} \equiv z_t = e_{19}$	task-completion claim
	$\mathit{verify} \equiv s_t = s_{11}$	verification / confirm phase

Specifications. G, F, X, U, W denote always, eventually, next, until, and weak until. Table 29 lists the six specifications with their model-checked and observed violation rates.

Product and queries. The product $M \times \mathcal{M}_\varphi$ has states $(s, u) \in Q \times U$ and transitions $(s, u) \xrightarrow{z} (\delta(s, z), \Delta(u, z, \delta(s, z)))$; only its reachable part is kept. Replaying the calibration split on the product, with absorbing outcome states FAIL and SUCC entered from the last non-terminal state, yields a discrete-time Markov chain on which we evaluate PCTL [14] queries:

$$\mathbb{P}_{=?}\big[\mathsf{F}\ \mathit{bad}\big] \text{ at } (s_0, u_0) \qquad \text{population probability of violating } \varphi, \tag{13}$$

$$\mathbb{P}_{=?}\big[\mathsf{F}^{\le H}\ \mathit{FAIL}\big] \text{ at } (s, u) \qquad \text{specification-refined risk, used as a prefix score.} \tag{14}$$

The same monitor also runs online: a prefix raises an alert at step t iff $u_t = \mathit{bad}$, i.e. iff it is a bad prefix of φ.

Results. Table 30 reports, for each specification, the model-checked violation probability (13), the empirical violation rate on failed and successful test runs, the held-out AUPRC of the refined risk (14) against the plain readout $\rho(s)$, and the precision and recall of the bad-prefix alert alone. The model-checked probability matches the prior-weighted calibration rate by construction (e.g. .152 vs. .152 for χ_2). On held-out test runs, the six marginal violation rates differ from model queries by .002–.011 after weighting the outcome-conditional rates by the test failure prevalence. This comparison evaluates the six specified event rules. The overall violation rates weight the failed- and successful-run rates by their test counts: 365/718 for τ^2-Bench and 2,318/1,122 for TerminalBench. On τ^2-Bench, the monitor for χ_2 distinguishes three histories within s_9: ready, awaiting a read, and violated. Their nonterminal calibration visit counts are 283, 53, and 69; fitted three-step failure probabilities are .097835, .157518, and .307722, respectively. The shared DFA readout is .138457 for all three, and the paired-state score raises nonterminal test AUPRC from .457446 to .510213. Ready means that no violation has occurred and either no write has occurred or a read has followed the latest write. Awaiting a read means that an unviolated prefix ends with a pending write; other event classes

Table 29: **Abstract-event specifications and violation queries.** Hand-written safety properties over the labeled events of τ^2-Bench (H=3, seed 13) and TerminalBench (H=3, seed 7); each monitor automaton has two to four states. Rates are percentages. Model uses calibration-fitted transitions; Test is the observed rate over all test trajectories. Failed and Successful give test violation rates conditional on the terminal outcome. Analyst-selected requirements are instantiated over manually labeled events or states, with one H=3 graph per benchmark; the readings describe these abstract labels.

		All trajectories		Test by outcome	
Spec.	LTL_f formula and reading	Model	Test	Failed	Successful
τ^2: χ_1	$\neg write\ \mathsf{W}\ detail$ No write before a detail lookup	18.7	17.8	34.5	9.3
χ_2	$\mathsf{G}\neg(write \wedge \mathsf{X}(\neg read\ \mathsf{U}\ write))$ No two writes without a read in between	15.2	16.1	35.3	6.3
χ_3	$\mathsf{G}\neg(write \wedge \mathsf{X}write \wedge \mathsf{XX}write)$ Never three writes in a row	3.6	4.7	10.4	1.8
Terminal: ψ_1	$\mathsf{G}\neg(warn \wedge \mathsf{X}warn)$ Never two format warnings in a row	12.8	13.9	18.0	5.3
ψ_2	$\neg\mathsf{F}(warn \wedge \mathsf{XF}(warn \wedge \mathsf{XF}\ warn))$ At most two format warnings in total	10.4	11.4	15.1	3.9
ψ_3	$\neg complete\ \mathsf{W}\ (verify \wedge \neg complete)$ No completion claim before verification	8.3	8.5	11.6	2.1

G, F, X, U, and W denote always, eventually, next, until, and weak until, respectively.

leave that memory unchanged. The violated state persists after any later event, including a read. On TerminalBench, adding the consecutive-warning rule changes AUPRC only from .1793 to .1812. For the total-warning rule, some paired states have lower fitted risk after violation; these estimates do not support treating every warning history as uniformly high risk.

Recorded examples for the rule memory. The three examples in Table 2 are calibration prefixes from different tasks, ending at s_9. Its visit counts cover all nonterminal calibration visits to each paired state, not just the selected example. For each rule state, we selected the shortest such prefix, breaking ties by trajectory ID; the ready example additionally requires a preceding write. Selection did not use outcomes or risk estimates. The event sequences and their recorded DFA paths (including initial state s_0) are:

$$\begin{aligned} \text{Ready:} \quad & (e_3, e_{15}, e_0, e_5, e_7) && \mapsto (s_0, s_0, s_4, s_4, s_7, s_9), \\ \text{Awaiting a read:} \quad & (e_{12}, e_2, e_{10}, e_5, e_0) && \mapsto (s_0, s_0, s_0, s_3, s_1, s_9), \\ \text{Violated:} \quad & (e_0, e_5, e_{12}, e_{10}, e_5, e_0) && \mapsto (s_0, s_0, s_3, s_4, s_3, s_1, s_9). \end{aligned}$$

The extraction replays every calibration transition and reproduces all three visit counts before selecting these examples. The violated example first enters the bad monitor state at step 5, in DFA state s_1, and subsequently reaches s_9. Its first six recorded tools are `reset_apn_settings`, `reboot_device`, `toggle_data`, `enable_roaming`, `can_send_mms`, and `check_wifi_calling_status`. Both `reboot_device` and `can_send_mms` map to e_5; the latter returns that the messaging app cannot send MMS messages. The replay establishes a violation of χ_2 over the learned alphabet and shows how the two operations share an abstract event.

F.2 Process-Conformance Audit

Scope and provenance. Section 4 derives five policy- or tool-grounded specification families from the 94 failures and audits all 359 retail executions aligned to the frozen H=3, seed-13 DFA. Definitions were recorded before whole-population counting, all five were retained, and original calibration/test membership and terminal outcomes were unchanged. This is a retrospective audit of the same executions used for requirement discovery.

Monitors and product exploration. Dedicated deterministic monitors implement the φ_1–φ_5 schemas in Section 4; Table 31 gives the grounded safety formulas. Semantic review narrowed φ_2 after confirming the same-SKU exchange exception; superseded results are not used. For each order/item/payment binding, the monitor checks the current request against preceding source facts

	$\mathbb{P}[\mathsf{F}\ bad]$	Violation rate (test)		AUPRC		Bad-prefix alert	
		failed	successful	$\rho(s)$	refined	precision	recall
χ_1	.187	.35	.09	.457	.489	.25	.33
χ_2	.152	.35	.06	.457	.510	.43	.31
χ_3	.036	.10	.02	.457	.477	.45	.08
ψ_1	.128	.18	.05	.179	.181	.14	.16
ψ_2	.104	.15	.04	.179	.196	.12	.14
ψ_3	.083	.12	.02	.179	.188	.17	.08

Table 30: Specification checking on the product chain, non-terminal test prefixes, $H{=}3$, one seed per benchmark. "Refined" is the PCTL score (14) on the product; the bad-prefix alert is the monitor alone.

before its reply updates memory, and missing evidence remains unknown. Each audited nonterminal step contributes an annotated frozen-DFA edge (s, e, L, s'). Exhaustive reachability over each DFA–monitor product covers the finite empirical graph: these five retail properties have 364, 28, 196, 268, and 312 reachable product states, respectively, and all five reach bad states. Source calls and replies validate the concrete violations reached in these products, whose empirical edges can recombine into unrecorded paths.

Table 31: **Tool-grounded retail specifications φ_1–φ_5.** Safety formulas as compiled by FLLOAT; propositions are evaluated per tool call against the latest preceding tool-confirmed facts for the bound order, item, or payment method. $known$ holds when that evidence exists; the companion evidence property $\mathsf{G}(call \rightarrow known)$ separates unknown from satisfied.

	Safety formula	Reads as
φ_1	$\mathsf{G}\big((attempt \wedge known) \rightarrow status_ok\big)$	the last confirmed status of the order permits the operation
φ_2	$\mathsf{G}\big((replace \wedge known) \rightarrow equal\big)$ $\wedge\ \mathsf{G}\big((modify \wedge known) \rightarrow changed\big)$	replacement lists have equal length; pending modifications change the item
φ_3	$\mathsf{G}\big((item_call \wedge known) \rightarrow quantity_ok\big)$	requested quantity does not exceed the order's confirmed count
φ_4	$\mathsf{G}\big((replace \wedge known) \rightarrow available_variant\big)$	replacement is an available variant of the same product
φ_5	$\mathsf{G}\big((return_call \wedge known) \rightarrow (original \vee gift)\big)$	refund goes to the original payment or the owner's gift card

Verification. Nine unit tests cover extraction and monitor boundaries. An independent implementation reconstructs source predicates for all 359 executions, checks 1,575 rule-call decisions and aggregate verdicts, and explores all five reachable product graphs. Separately, FLLOAT version 0.3.0, an LTL_f-to-DFA library,[7] compiles one safety and one evidence-completeness automaton per rule after grounding data bindings to Boolean propositions. The ten automata reproduce all 1,575 call decisions and 1,795 rule/trajectory verdicts with zero mismatches. Safety detects known violations; evidence completeness distinguishes unknown from satisfied when there is no violation.

F.3 Task-27 Temporal Ordering Example

Selection and requirement provenance. The task-27 case illustrates the retail property φ_1. It was selected retrospectively by inspecting tool precondition errors in the frozen test split, then retrieving the other test execution of that task. The failed execution uses `gpt-4.1-2025-04-14`; the successful execution uses `o4-mini-2025-04-16`. Both are trial 0 under the retail default policy, with the same user-model configuration. The pair illustrates how different operation histories affect the task requirement. The requirements come from the original task instructions and recorded runtime policy. For target order `#W7181492`, the user requests returns of a hose and backpack and an exchange of hiking boots to a waterproof variant, preferring the exchange if both requests cannot be fulfilled. The policy restricts returns and exchanges to delivered orders and changes the status to `return requested` or `exchange requested` upon acceptance, respectively. The recorded task provides no operation to restore delivered status after either request.

[7] `https://github.com/whitemech/flloat`

Propositions and monitor. R_o and E_o mark calls to the return and exchange tools with target order o, including calls that are rejected. R_o^+ and E_o^+ additionally require a structured tool response confirming the matching order and the corresponding requested status; an error response never establishes acceptance. B_o^+ requires E_o^+ with original item `8118291112`, waterproof replacement `8277474082`, and the task's payment argument. Both observed exchange calls target these boots, but only the successful execution receives acceptance. The monitor stores four Boolean flags: a prior accepted return, a prior accepted exchange, and persistent violation flags for φ_{RE} and φ_{ER}. For each step, an exchange attempt after a prior accepted return sets the first violation flag; a return attempt after a prior accepted exchange sets the second. Acceptance flags are then updated from the current response. This order implements the strict later-step relation XF in Table 32. An accepted operation at the last position alone does not violate either safety property.

Finite-state exploration and concrete validation. Each observed nonterminal step contributes an edge (s, e, L, s') with the original transition $s' = \delta(s, e)$ and tool-grounded label L. We retain each occurrence's tool-grounded label when multiple calls share a DFA edge. The graph is the union of these annotated edges from the two executions, with initial state s_0. Composing it with the monitor and exhaustively exploring reachable paired states yields 19 states and reaches both violation flags. All 30 recorded nonterminal transitions match the frozen $H{=}3$, seed-13 DFA. The product graph can recombine observed edges; source messages and tool replies validate the recorded first violating prefixes, of lengths 12 and 16. The DFA, outcome labels, and data splits remain fixed.

Table 32: **Temporal requirements for retail task 27.** Failed: GPT-4.1; successful: o4-mini. V: first violation step; +: tool-confirmed acceptance.

LTL_f specification	Failed	Successful
$\varphi_{RE} = \mathsf{G}\neg(R_o^+ \wedge \mathsf{XF}E_o)$	V: 12	Holds
$\varphi_{ER} = \mathsf{G}\neg(E_o^+ \wedge \mathsf{XF}R_o)$	Holds	V: 16
$\mathsf{F}B_o^+$	Unmet	Step 13

Error mechanism and task outcome. In the failed execution, step 8 reports the order as delivered; the return accepted at step 11 changes it to return requested. The exchange at step 12 is rejected because the order is no longer delivered, establishing the first violation of φ_{RE}. The successful execution accepts the requested exchange at step 13; its subsequent return at step 16 is rejected for the corresponding status precondition, violating φ_{ER}. Both runs pass the official action-matching checks, including the exchange-call check; tool responses determine whether each requested operation was accepted. Only the successful run passes the database comparison; the final rewards are 0 and 1, respectively. Thus, tool feedback identifies the local error mechanism, and the task's exchange preference explains why the blocked operations have different implications for completion.

Independent checks. The replay was checked against the original assistant actions, tool replies, and official reward records, reproducing the transition, acceptance, and violation steps.

F.4 Task-Grounded Requirements and Counterexample Validation

An additional completion-property case uses retail task 103, selected from an eight-pair execution review. That review selected same-task, same-configuration success/failure pairs by shared raw-operation and event-prefix lengths, with deterministic ID tie-breaking; risk scores and alerts were not selection criteria. Both executions are from the frozen test split, with agent `gpt-4.1-2025-04-14`, the retail default policy, and the same user-model configuration; they are trials 0 and 1 of the same task. The case uses the frozen splits, labels, and DFA.

Table 33: **Task requirements and their satisfaction in paired executions.** The two test executions share retail task 103 and the agent configuration. Entries give the step at which a successful target operation satisfies the property; "Unmet" means absent at termination. F means eventually within the completed finite execution.

LTL_f specification	Task requirement	Failed run	Successful run
$\mathsf{F}r_b \wedge \mathsf{F}r_j$	Request bookshelf and puzzle returns	Unmet	12
$\mathsf{F}r_k$	Request backpack return	8	13
$\mathsf{F}a$	Apply requested shipping address	19	16
$\mathsf{F}c$	Change the specified item to red	20	19

Task propositions. The target return order is `#W6239298`, containing bookshelf item `4900661478` and puzzle item `3614853563`. For r_b and r_j, we require `return_delivered_order_items` with the task's order, item, and payment arguments, together with a returned order whose status is `return requested` and whose `return_items` contains the corresponding item. The same checks define r_k for backpack item `7824298782` in order `#W9218746`. For a and c, the target is order `#W4860251`: the call must match the requested address or item replacement, and the response must contain that address or replacement item `8964750292`. All arguments are matched against the task's evaluation criteria, with item lists compared independently of their ordering. These propositions are derived from actions and tool feedback; outcome labels and official check results are used only for subsequent validation.

Task-labelled model and property monitor. Each recorded nonterminal transition contributes an edge (s, e, L, s'), where $s' = \delta(s, e)$ is its original DFA transition and $L \subseteq \{r_b, r_j, r_k, a, c\}$ is its tool-grounded proposition label. Parallel occurrences of the same DFA edge retain their individual tool-grounded labels L. We use the union of these edges from the two task executions and permit an explicit END transition from their observed final nonterminal states, s_8 and s_3. END records termination independently of success or failure; the final outcome-bearing model input is excluded. The property monitor stores the set B of obligations already fulfilled, with update $B' = B \cup L$. On finite paths ending in END, violation of the conjunction in Table 33 reduces to reachability of END with $B \neq \{r_b, r_j, r_k, a, c\}$. Exhaustive exploration of the task-labelled graph and monitor reached 64 paired states before END and found violating terminal paths. Source messages and tool replies validate the complete recorded failed path as a concrete counterexample.

Recorded counterexample and successful comparison. At step 12, both executions reach s_4. The failed execution calls `get_order_details` for `#W6239298`; the successful execution submits its return request and receives confirmation for both target items. The failed path ends at s_8 with $B = \{r_k, a, c\}$; the successful path ends at s_3 with all five propositions fulfilled. The official action checks match the four property outcomes in Table 33; both tracking-number checks pass, while only the successful run passes the database comparison. The incomplete obligation set and source feedback identify the omitted target return request.

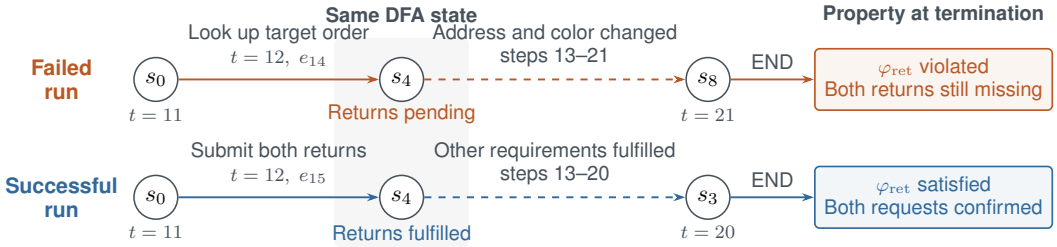

Figure 11: **Requirement checking on PrefixGuard replay paths.** Selected steps from the same-task retail pair (H=3, seed 13). Both paths reach s_4 at step 12, but the property monitor distinguishes a lookup from confirmed return requests. The failed path ends with $\varphi_{\text{ret}} = \mathsf{F} r_b \wedge \mathsf{F} r_j$ unmet; both runs satisfy the other operation requirements. Dashed arrows omit intermediate DFA transitions; END is an added termination marker.

Independent checks. Every replayed transition, source message, tool response, and reward record was checked against the property monitor; the recorded satisfaction steps were reproduced. Order lookups, wrong target orders, and unsuccessful return responses were excluded as evidence of a completed return request.