AFR: ACTION-LEVEL FEEDBACK AND CROSS-TRIAL MEMORY FOR FROZEN LANGUAGE-MODEL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models are increasingly deployed as agents in interactive environments, where failed actions are frequent. However, two limitations persist. The environment's reply to a failed action is typically a terse no-effect message that carries no repair cue, so the agent repeats state-bound mistakes across trials. Existing remedies entangle two uses of failure information, blocking predicted-to-fail actions before execution and accumulating reflection memory across trials, leaving their individual contributions unclear. To address these issues, we propose AFR, an external interface that decouples the two channels for a frozen agent. Specifically, a state tracker feeds a failure gate that intercepts predicted-to-fail actions and emits a state-bound reason with a repair suggestion, while a memory channel distills each failed trial into lessons injected into later trials. Both attach to the actor at inference time without fine-tuning. On 134 ALFWorld tasks with Qwen3-8B under approximately matched cumulative call budgets, each channel helps on its own: AFR improves success from 61.2% for the bare actor to 88.8%. In a separate evaluation batch, replacing the rule gate with an oracle that consults the environment's admissible-command list increases success from 73.1% to 76.9%, with uncertainty spanning no improvement, while the gate-plus-memory agent reaches 90.3%, above the oracle gate. In a pre-registered transfer test on a second suite, the gate shows no detectable improvement while memory remains beneficial, and a learned-gate checkpoint transfers across actors. AFR reduces avoidable action failures at execution time and reduces repeated failures across trials, without updating the underlying language model; these mechanism-level reductions help explain the success gains under the matched protocol. Code, rollout archives, and analysis scripts are available at `https://anonymous.4open.science/r/iclr2027-14BC`.

1 INTRODUCTION

Agents based on a large language model (LLM) interact with an environment by proposing actions, observing their consequences, and choosing what to do next (Yao et al., 2022b; Wang et al., 2024a). An action that is plausible for a task can nevertheless be invalid in the current state: an agent may try to take an object from a receptacle it has not reached, or heat an item it is not holding. In text-based environments such as ALFWorld (Shridhar et al., 2020), these different causes can produce the same terse no-effect response. Such feedback identifies an unsuccessful step without explaining which precondition was violated or how to repair the action. Repeated attempts then consume the interaction budget. The question is when to use failure information: before execution to prevent the current failure, or after a trial to prevent its recurrence. We study how an external interface can use this failure information at both timescales while keeping the actor's model weights fixed.

Failure information can enter the agent's decision process at different times. Before execution, a feasibility check can intercept a proposed action and return feedback grounded in the current state. After an unsuccessful trial, reflection can summarize the trajectory and preserve lessons for another attempt at the same task. Action-feasibility methods, such as SayCan (Ichter et al., 2023), and reflection-based agents, such as Reflexion (Shinn et al., 2023), motivate these intervention points. They address distinct decisions: whether to execute the current proposal, and what experience to carry into the next attempt.

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.

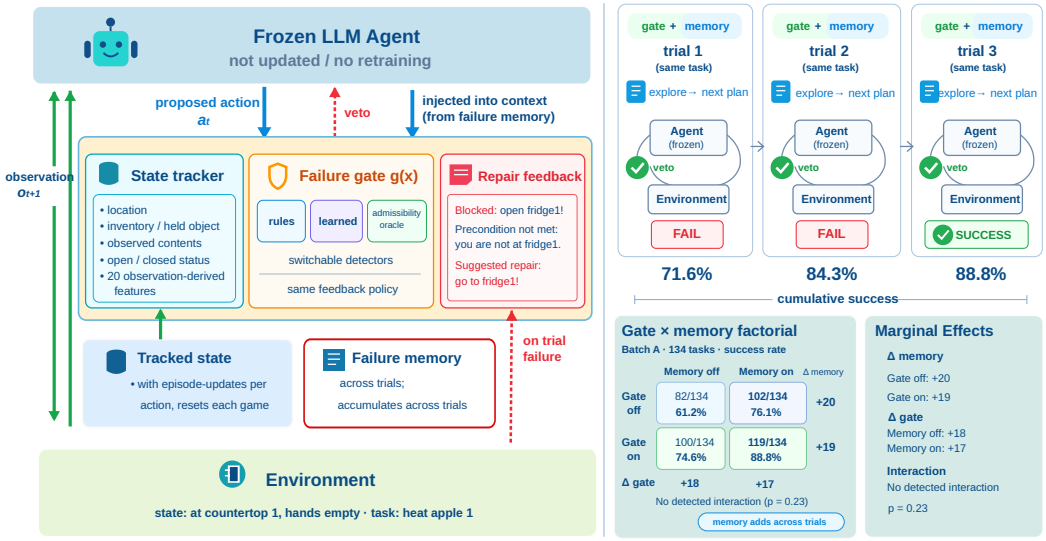

Figure 1: AFR uses failure information to guide a frozen agent's current actions and later attempts. Left: a state tracker supports a gate that blocks predicted-to-fail actions and returns repair feedback, while reflection memory carries lessons from failed trials into subsequent attempts. Right: the trial sequence illustrates this reuse, and the within-batch comparison shows that combining gating and memory improves success over either intervention alone.

Our central question is how action-level gating and cross-trial memory should be organized and evaluated within a frozen-agent pipeline. A gate may avoid an ineffective environment step, but its rejection also requires the actor to propose another action. Reflection can preserve experience across attempts, but it also consumes calls and adds context. Both mechanisms therefore need to be evaluated against a cumulative interaction budget. A related question concerns the gate itself: how does changing the criterion for rejecting an action affect a system that uses the same feedback-generation policy?

We introduce AFR, an external interface with separately configurable gate and memory channels (Figure 1). A state tracker maintains location, inventory, and observed receptacle contents from the visible interaction history. The gate inspects a proposed action against this tracked state. If the gate predicts failure, AFR withholds execution and returns a state-bound reason and a repair suggestion through a feedback formatter, with a generic fallback when no specific repair is available. The actor then re-proposes within a bounded retry loop. Executed actions produce observations that update the tracker. After an unsuccessful trial, the memory channel generates a textual reflection from the executed trajectory and inserts it into the context of a later trial. Gate messages are excluded from the reflection input, although gating can change the trajectory on which reflection operates. The interface changes the actor's inputs and action execution without fine-tuning the actor.

We evaluate this design by enabling gating and memory separately and together, alongside a bare-actor condition, on the same ALFWorld tasks with Qwen3-8B (Yang et al., 2025a). The protocol uses approximately matched cumulative call budgets and task-paired comparisons, with model calls, prompt tokens, and environment steps reported alongside task success. We also compare rule-based, learned, and admissibility-oracle gates under a shared reason-and-repair policy. This fixes the feedback-generation procedure while allowing gate decisions and subsequent trajectories to differ. The oracle provides a comparator with access to the environment's admissible-command list. That privileged information is withheld from the actor and state tracker. Cross-suite evaluation on ScienceWorld (Wang et al., 2022) and cross-actor reuse of a fixed learned gate examine the scope of the interface beyond its primary setting.

The work makes the following contributions:

- We propose AFR, an external interface that connects state-based action gating with cross-trial reflection memory while keeping the language-model actor frozen. The interface uses tracked observations to generate repair feedback for rejected actions and carries lessons from unsuccessful trials into later contexts. Its separately configurable channels support studying immediate action correction and experience reuse within the same pipeline.
- We design a controlled evaluation that enables gating and memory individually and jointly, alongside a bare-actor baseline. The comparison uses the same tasks and actor under approximately matched cumulative call budgets, accounts for re-query and reflection calls, and reports prompt-token and environment-step costs alongside task success. Task-paired analyses assess the individual contributions of the channels and their interaction.
- We investigate how gate criteria and deployment settings affect the interface through rule-based, learned, and admissibility-oracle comparisons under a shared feedback-generation policy. Cross-suite evaluation examines the dependence on environment-specific state checks, while cross-actor tests reuse a fixed learned-gate checkpoint without retraining. Together with trajectory-level diagnostics, these comparisons examine the roles of action rejection, repair feedback, and memory across settings.

2 Related Work

2.1 Language-Model Agents and Action Feasibility

Language models interact with environments through reasoning and tool use (Yao et al., 2022b; Schick et al., 2023; Wang et al., 2023; 2024a). ALFWorld, TextWorld, and related benchmarks make the gap between a plausible command and a successful trajectory measurable (Shridhar et al., 2020; Côté et al., 2018; Yao et al., 2022a; Liu et al., 2024). SayCan combines language-model task knowledge with skill affordances (Ichter et al., 2023); ProgPrompt exposes actions and objects through programmatic prompts (Singh et al., 2022). A closer family checks or repairs actions before or at execution: CAPE queries the model for corrective actions on precondition errors (Sundara Raman et al., 2024), DoReMi detects plan–execution misalignment and re-plans (Guo et al., 2024), and PRED revises plans from environmental feedback (Kim et al., 2025). These systems couple detector and repair loop into one pipeline; AFR instead holds the actor, task set, and budget fixed and measures what the execution-time gate-and-repair intervention is worth on its own, with memory, and against an oracle bound. An intervention returns a state-bound message to the same actor. Its focus is how state-bound feedback and cross-trial memory influence the use of predicted and realized failures.

2.2 Feedback and Self-Reflection

Reflexion, Self-Refine, and CRITIC use feedback to revise decisions (Shinn et al., 2023; Madaan et al., 2023; Gou et al., 2024). The effectiveness of self-correction depends on the feedback and setting (Huang et al., 2024; Kamoi et al., 2024). Prior comparisons place these methods against unassisted or differently configured baselines. Budgets, intervention times, and context management usually differ jointly. We measure within-episode vetoing and between-trial reflection in one framework, under one cumulative-call budget, with task-paired tests, and with a hybrid that applies both. This design estimates the individual and joint contributions of gating and memory, together with their interaction costs.

2.3 Experience Reuse and Predictive Interfaces

Failure experiences can support fine-tuning and memory construction (Song et al., 2024; Wang et al., 2024b; Yang et al., 2023). Search-based frameworks such as LATS fold action-feasibility estimates into lookahead tree search (Zhou et al., 2024); AFR prices the same information at execution time and asks what the detected failures are worth afterwards. ExpeL and similar experience-retrieval systems expose prior interactions to an agent (Zhao et al., 2024; Park et al., 2023; Packer et al., 2023; Wang et al., 2024c; Zheng et al., 2024; Kagaya et al., 2024; Zhang et al., 2024). World models predict transitions for planning (Ha & Schmidhuber, 2018; Hafner et al., 2020). AFR uses a narrower immediate-failure score and a separate language interface. Keeping this computation

outside the actor also distinguishes it from model editing (Meng et al., 2022; Yao et al., 2023b). Our oracle control compares rule-based gating with access to an admissible-command list under a shared feedback-generation policy.

A recent line of work also keeps the actor frozen and builds the adaptation outside it, differing in where the external knowledge comes from. AutoManual induces environment rules online with a dedicated rule-management agent (Chen et al., 2024), and CFGM grounds coarse-to-fine experience memory in an offline collection phase (Yang et al., 2025b). MACLA distills trajectories into procedural memory with Bayesian selection (Forouzandeh et al., 2026), AdaMEM adapts a trajectory memory at test time (Zhang et al., 2026), and ReflexGrad recovers within an episode through dual-process routing (Kadu & Krishnan, 2025). These systems evaluate each mechanism as a whole, reporting success with their own actor, split, and budget. AFR instead decomposes the failure-information interface into a now channel and a later channel under approximately matched budgets, and compares gate criteria using an admissibility oracle. Appendix Q tabulates the reported numbers.

3 METHOD

3.1 PROBLEM SETUP AND INTERFACE OVERVIEW

We consider a language-model actor interacting with a stateful environment to complete a task q. A trial is an attempt at that task, and a step is an action decision within a trial. The actor's parameters θ remain fixed throughout evaluation. AFR operates outside the actor, controlling which proposed actions reach the environment and which feedback enters the actor's context. Its gate and memory channels can be enabled independently or used together (Figure 2).

Let k index trials and t index proposal calls within a trial. We denote the visible environment history by $h_t^{(k)}$, the actor's full context by $c_t^{(k)}$, and the reflection cards available before a trial by M_k. The context contains the task, observations, prior interaction, available memory, and any interface feedback. The actor proposes an action, while the tracker and feature map construct its state-dependent representation:

$$a_t^{(k)} \sim \pi_\theta(\cdot \mid c_t^{(k)}), \qquad s_t^{(k)} = T(h_t^{(k)}), \qquad x_t^{(k)} = \phi(s_t^{(k)}, a_t^{(k)}). \tag{1}$$

A rejected proposal changes the actor's context through feedback but does not change the environment. An executed action produces an observation that updates the tracker. Memory supplies information across trials, while the tracker represents the current environment state within a trial.

3.2 OBSERVATION-BASED STATE TRACKING

The tracker maintains a structured estimate of the state from visible observations and executed-action feedback. In ALFWorld, this includes the current location, held object, known receptacles, observed contents, and opening or closing feedback. It initializes from the environment's initial observation on reset and updates these records after execution. The state-action feature map combines these records with the proposed command template. The complete feature definitions and unknown-value encoding are in Appendix A.

The tracked state reflects what has been observed and may be incomplete or incorrect. In particular, an unobserved object need not be absent from the environment. The tracker has no access to expert plans, evaluator success flags, or the simulator's admissible-command list. These information restrictions also apply when a privileged oracle is used as the gate.

3.3 ACTION-LEVEL GATING AND REPAIR

The gate and feedback formatter have separate roles:

$$v_t^{(k)} = g(x_t^{(k)}) \in \{0, 1\}, \qquad d_t^{(k)} = D(s_t^{(k)}, a_t^{(k)}), \tag{2}$$

where $v = 1$ denotes a veto. When $v = 0$, the interface executes the action and passes the resulting observation to the actor and tracker. When $v = 1$, it withholds the action and returns $d_t^{(k)}$: a state-bound reason and repair suggestion when a rule applies, or a generic rejection otherwise. The actor

then proposes another action from the augmented context. Every proposal, including a re-query, consumes a model call, whereas only executed commands consume environment steps.

The local veto loop permits at most three re-queries. If no proposal is accepted before this limit, the interface skips execution and supplies a fixed message asking the actor to re-plan from the current state (Appendix A.6). Local retry exhaustion does not by itself terminate the task or execute a no-op. Further proposals remain subject to the trial's stopping and budget conditions.

We instantiate the gate with rules, a learned predictor, or an admissibility oracle. The rule gate checks command syntax and state-dependent preconditions, including location, inventory, and object presence. The learned gate maps observation-derived state-action features to a prediction of the immediate no-effect response. Its weights are fixed at evaluation. The oracle checks the proposed command against the simulator's admissible-command list using the implementation's command matching procedure. This list is available only to the oracle gate. Admissibility does not establish that an executable action advances the task, so this comparator provides no general upper bound on task success. All gate variants use the same formatter and retry policy. This holds the feedback-generation procedure fixed. Different veto decisions can still produce different messages, contexts, and trajectories.

3.4 Cross-Trial Reflection Memory

Following Shinn et al. (2023), the memory channel summarizes an unsuccessful trial into a textual lesson for a subsequent attempt. Its input includes the task's initial observation, existing reflections, and the executed action-response trajectory τ_k^{exec}:

$$r_k = R(o_0^{(k)}, M_k, \tau_k^{\text{exec}}), \qquad M_{k+1} = U(M_k, r_k), \tag{3}$$

where R is the reflection prompt and U adds the resulting card to the retained memory collection. The implemented memory channel retains at most three cards and injects them into later trial contexts. Generation and text-normalization limits are specified in Appendix A.6.

Rejected proposals and gate repair messages are excluded from the reflection trajectory. The channels nevertheless interact through execution: gating can change the actions and observations from which a lesson is written. Enabling only the gate gives action-level intervention. Enabling only memory gives reflection-based retries, and AFR enables both. With no prior-trial cards, the initial AFR trial uses the gate alone, and later attempts also receive accumulated lessons. Algorithm 1 summarizes this inference flow. Trial scheduling and call limits are specified in Section 4.1.

4 Experiments

4.1 Experimental Setup

The primary ALFWorld comparisons use the same 134 `valid_unseen` tasks with Qwen3-8B (BF16, greedy decoding, thinking disabled, at most 32 new tokens per proposal, and seed 42 reset per episode). The matched-feedback controls use 50 or 150 model calls and 50 executed actions. Reflection arms run up to three trials with a 50-call proposal cap per trial and separate reflection calls. The resulting approximately 152-call ceiling is therefore only approximately matched to the 150-call conditions. Costs include reflection calls and are reported as calls, prompt tokens, and environment steps. Full implementation and prompt details appear in Appendix A.

For statistical analysis, we use exact two-sided McNemar tests for paired success comparisons and Newcombe confidence intervals for paired differences in proportions. The reflection family comprises six primary contrasts and the gate and memory factorial, with Holm correction within family. Non-significance is not interpreted as equivalence, and a confidence-interval endpoint is not a theoretical bound on achievable performance. The gate and feedback factorial uses a closed analysis family. The reflection family was designated after its runs and is disclosed as post hoc. The oracle family was frozen before oracle results were inspected. Cross-suite tests and other secondary analyses are reported with their family designations. The same-batch AFR rerun for the oracle comparison was approved and launched before its results were seen. We retain the stated primary tests, including the ScienceWorld gate comparison (Section 4.2).

Algorithm 1 AFR inference flow for a task

Require: Frozen actor π_θ, tracker T, gate g, formatter D, reflection prompt R, trial and budget schedule

Start with no prior-trial reflection cards
for each scheduled trial k **do**
  Reset the environment; initialize tracker and actor context with M_k
  Initialize the executed trajectory τ_k^{exec}
  while the trial remains active and proposal budget is available **do**
    Propose an action; apply g and the bounded repair loop using D
    Count the initial proposal and every re-query as model calls
    if an action is accepted and execution is permitted **then**
      Execute it; append its response to τ_k^{exec}
      Update tracker and actor context; count an environment step
    else
      Leave the environment unchanged and retain re-planning feedback
    end if
  end while
  if the task succeeds **then**
    Stop further trials
  end if
  if a reflection-backed retry is scheduled **then**
    Generate r_k from initial observation, M_k, and τ_k^{exec}
    Count the reflection call and update retained cards for the retry
  end if
end for

All results come from three frozen-protocol rollout batches (Table 2). Greedy decoding is deterministic per GPU model but not across GPU models. Consequently, all headline contrasts are within-batch and task-paired. We do not subtract success rates across batches. Within the ALFWorld batch used for the factorial comparison, the Bare trajectories are identical at the 50- and 150-call caps on all 134 games. The initial gated trial also reproduces the corresponding 50-call gate condition on all 134 games. These checks underlie the archived-prefix budget analysis reported below.

We organize the experiments around four questions: whether gating helps without memory, whether memory helps without gating, whether the channels are complementary under one protocol, and how gate criteria and deployment settings affect performance across budgets, environments, and actors.

4.2 Quantitative Results

We first isolate the two intervention channels in a 2×2 design. The four cells enable action-level gating and cross-trial memory independently and jointly while holding the actor, task set, and cumulative interaction protocol fixed. The two single-axis conditions enable only one channel, and AFR enables both. Table 1 combines our ALFWorld batch results with reported Qwen3-8B comparisons. AFR reaches 90.3% in batch B, its best reported batch. The literature rows provide same-backbone context, with different interaction budgets and repetition protocols.

The factorial model estimates positive main effects for both factors and does not detect a significant interaction ($\beta = +0.29$, $p = 0.23$). We therefore describe the channels as complementary under this protocol, without interpreting a null interaction as proof of exact additivity or equivalence. The shared-feedback control further allows the gate identity to be compared while holding the reason-and-repair formatter fixed. Those exploratory contrasts are reported with the matched-feedback audit.

A stronger gate criterion may reject more inadmissible actions, but each veto also consumes a re-query. We therefore compare the rule gate with an admissibility oracle under the same feedback policy and budget. In batch B, the rule gate reaches 73.1% and the oracle reaches 76.9%. The paired difference is 3.7 percentage points with a Newcombe 95% confidence interval of $[-2.0, +9.5]$ (Table 1, batch B rows). The oracle vetoes more often and incurs more local re-query exhaustion, illustrating the cost of using a more aggressive criterion.

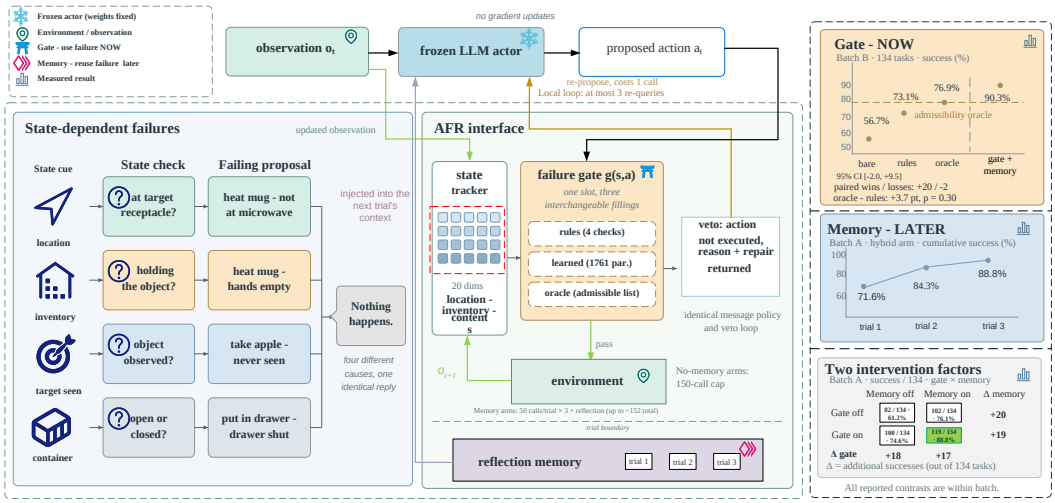

Figure 2: AFR pipeline: state-dependent failures (left), action gating and cross-trial memory (middle), and evaluation summaries (right). The gate comparison uses batch B. The cumulative trial curve and gate and memory factorial use batch A.

Table 1: ALFWorld results with Qwen3-8B. Block A: paired comparisons against Bare (Δ: percentage points, p: exact two-sided McNemar test). Block B: paired comparisons against the preceding row. Costs are per-task means including reflection. Block C lists reported same-backbone results from Kadu & Krishnan (2025), Table 3 (mean ± SD, ten seeds, 15-step protocol); AFR reports both of its batches.

Method	Success (%)	Δ (pp)	Wins/losses	p	Calls	Tokens
Block A: our runs, ALFWorld 134 tasks, paired with Bare						
Bare	82/134 (61.2)	—	—	—	29.3	35.1k
Rules gate	100/134 (74.6)	+13.4	28/10	.005	31.9	49.3k
Learned gate	98/134 (73.1)	+11.9	26/10	.011	32.5	50.8k
Reflection memory	102/134 (76.1)	+14.9	20/0	1.9×10^{-6}	57.9	76.1k
AFR	**119/134 (88.8)**	**+27.6**	38/1	1.5×10^{-10}	43.9	57.7k
Block B: our runs, ALFWorld 134 tasks, gate-capacity ladder						
Bare	76/134 (56.7)	—	—	—	30.4	36.6k
Rules gate	98/134 (73.1)	+16.4	32/10	.0009	34.0	55.8k
Oracle gate	103/134 (76.9)	+3.7	10/5	.30	35.7	67.6k
AFR	**121/134 (90.3)**	**+13.4**	20/2	1.2×10^{-4}	41.8	54.0k

Block C: reported results, same backbone (Qwen3-8B)

Method	Tasks	Demos	Success (%)	Source
Self-Refine (Madaan et al., 2023)	134	1	68.7 ± 1.9	Reported
ToT (Yao et al., 2023a)	134	1	69.7 ± 2.2	Reported
LATS (Zhou et al., 2024)	134	1	72.7 ± 2.0	Reported
ReflexGrad (Kadu & Krishnan, 2025)	134	0	75.4 ± 2.2	Reported
AFR (ours)	134	1	**90.3**	Batch B, single run

Table 2: Evaluation batches and their roles. All contrasts are task-paired within batch. Bare baselines are not compared across batches.

Batch	Hardware	Contents	Bare	Used for
A	L20	2×2 + reflexion (ALFWorld)	82/134	2×2, reflexion contrasts (Table 1)
B	A100	oracle gate + AFR rerun (ALFWorld)	76/134	oracle comparison (Table 1)
C	A100	four arms (ScienceWorld)	8/75	cross-suite (Table 5)

The oracle is an operational comparator for admissibility-based detection, not a universal upper bound on task success: an admissible action can still be unhelpful, and a veto does not create unlimited budget. The same batch's gate-plus-memory condition reaches 90.3% and exceeds the oracle-only condition in a paired comparison, while using fewer prompt tokens. This comparison shows the value of the hybrid beyond the oracle-only comparator under this protocol. It does not identify the entire gap as a single causal memory effect.

We next test whether the ALFWorld-derived state checks transfer to ScienceWorld. The rule gate does not show a detectable improvement over the Bare condition, whereas reflection memory remains beneficial (Table 5). Because this suite is smaller and is underpowered for small effects, the result is evidence about portability rather than proof of a zero gate contribution. To assess cross-actor reuse, we apply the learned gate checkpoint to ChatGLM3-6B without retraining. The test asks whether the external gate artifact can operate with the same feature vocabulary around another actor. The observed improvement is a within-environment-family transfer result. It does not establish transfer across environment classes. The low-performing Llama floor case is reported in Appendix C because effects are not interpretable when the actor rarely completes the task.

4.3 Qualitative Visualization

Aggregate success rates do not show how the two channels interact during a task. Figure 3 follows one ALFWorld task across successive trials. The gate intercepts an action whose state precondition is violated and returns a repair instead of executing that proposal. Reflection then carries information from failed trials into a later context, where the search trajectory changes. The memory-only comparison illustrates the contrasting failure mode: generic no-effect feedback can be repeated without changing the search location. This example is qualitative. The quantitative comparisons are given in Table 1.

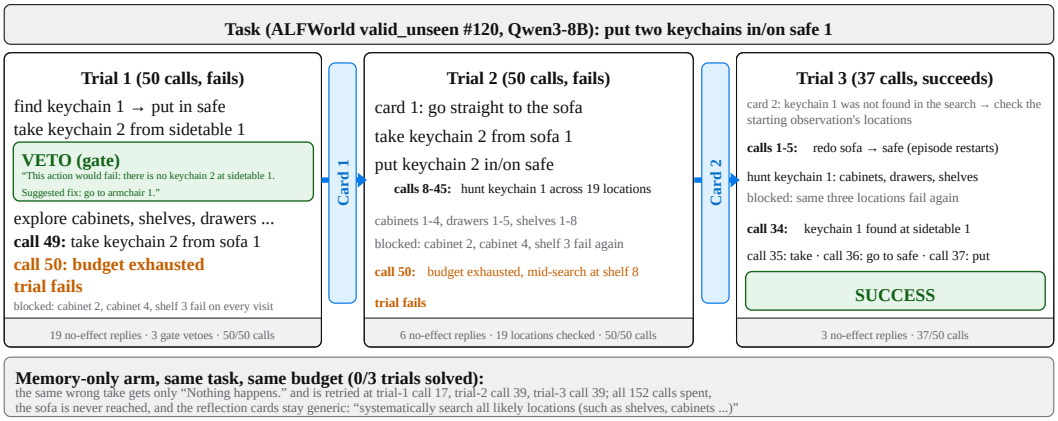

Figure 3: Qualitative ALFWorld case showing gate repair and cross-trial reflection. AFR reaches the task goal in its third trial after 37 calls. Memory alone repeats a failing action under the same budget. Excerpts are from evaluation logs. Ellipses mark omitted steps.

4.4 Ablation and Diagnostic Analysis

We next examine the interaction costs of the two channels. Figure 4 reconstructs success at each cumulative call budget from deterministic trajectory prefixes in batch A. Gating produces an earlier improvement in the budget curve, whereas memory gains require later trials. The hybrid combines these temporal patterns. The curves are archived-prefix analyses, not independent reruns at every budget level. Table 7 in Appendix E gives the per-trial cost decomposition. Relative to memory alone, AFR uses fewer calls, prompt tokens, and environment steps in the matched batch. These costs include re-queries and reflection calls. They should not be interpreted as an across-batch comparison with the Bare baseline.

The rules, learned, and oracle gates share the same reason-and-repair formatter and local retry policy. The matched-feedback audit in Appendix F separates detector identity from feedback content.

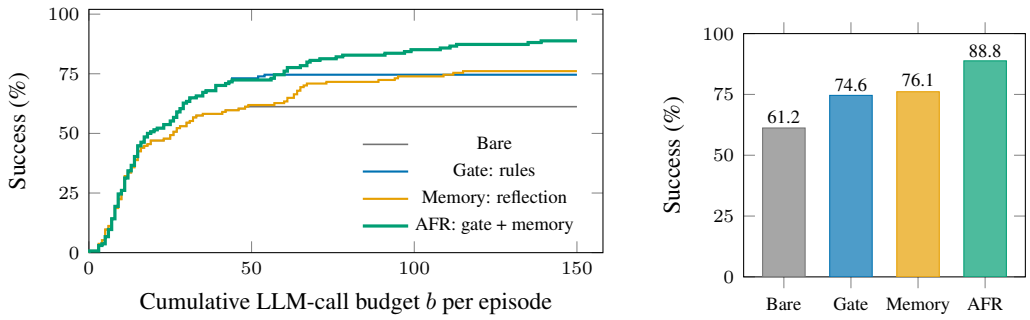

Figure 4: Left: success versus cumulative model-call budget, reconstructed from deterministic trajectory prefixes in the batch-A archives (Section 4.1). Right: end-point success of the four arms in batch A (Table 1, batch A rows).

Appendix E reports veto frequency, repair landing, local-loop exhaustion, trial rescue, and reflection-card production. Later-trial rates are conditional on earlier failure and use different task sets. The oracle lands more first repairs but also exhausts more local veto loops, showing why stronger admissibility detection need not yield higher task success under a finite re-query budget.

4.5 Analysis and Scope

The deployed gate bundles three components: the detector, the state-bound repair language, and the re-query that each veto buys. The shared formatter holds the language fixed across detectors; the matched-feedback audit isolates the language contribution and the batch B ladder bounds the detector contribution, and we do not claim to separate the remaining bundle further. What the bundle delivers under this protocol is the redirection of the actor's budget through rejected proposals rather than detection precision.

The transfer results scope the two channels. The learned-gate checkpoint reuses a feature vocabulary and parser around a second actor within the same environment family. The rule gate's knowledge does not cross environment classes, whereas reflection memory, which arrives as text and requires no engineered state, remains beneficial on ScienceWorld. This pattern is consistent with deployment-time knowledge living in the environment's vocabulary and run-time reflection in the agent's experience. Under the studied protocol, these results favor deploying the lower-cost intervention first and adding memory when multi-trial interaction is available, while further gate investment beyond the rule-based detector shows the smallest measured return (Table 1).

The pre-registered outcome is task success; the archives add a mechanism reading. In batch A, the share of executed actions that return the no-effect response is 33.1% for Bare and 36.1% for memory alone, but 15.3% with the rule gate and 17.6% for AFR, and the mean count of repeated no-effect actions per episode (pooled across trials) falls from 6.57 (Bare) to 5.46 (AFR). Fewer invalid actions and fewer avoidable repeats are the mechanism side of the success gains. A retry-without-reflection control is analytically degenerate here: a second trial without injected reflection has the same context as the first by construction, so under greedy decoding it replays the first trial's trajectory, and the memory arm's gain is attributable to the reflection text; the bare arm's identical outcomes at the 50- and 150-call caps ($134/134$ games) confirm the underlying determinism empirically.

5 Conclusion

We presented AFR, an external interface that lets a frozen language-model agent use failure information at two timescales. Across our evaluations, action-level repair and cross-trial memory improve task handling, while the admissibility oracle incurs additional re-query costs with an uncertain advantage over rules. Transfer results indicate that state-based gates depend on an environment's observable vocabulary, whereas experience reuse can be evaluated separately from engineered predicates. These findings support evaluating failure-handling interfaces by trajectories and interaction costs, alongside detector precision.

Reproducibility Statement

The main comparisons use the three evaluation batches of Table 2. Full rollout archives, source, preflight reports, and analysis scripts are released. The archives reproduce deterministically on their hardware (within-batch reruns of disjoint cells agree on $134/134$ trajectory prefixes), and pre-registered family assignments and protocol freezes are recorded in versioned protocol files accompanying the package. Implementation details, rollout signatures, and a discussion of limitations are in Appendices A, B, and H.

AI Use Statement

Generative AI assisted methodological review, code and trajectory inspection, experiment execution and monitoring, statistical analysis, figure drafting, and manuscript writing and revision. All reported numbers are computed from stored rollout archives and independently re-verified. The rewrite adds no unevaluated claims about new rollouts. This assistance is distinct from the language models evaluated as actors. The authors are responsible for verifying the final text, results, code, disclosures, and submission-policy compliance.

References

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. Automanual: Constructing instruction manuals by LLM agents via interactive environmental learning. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, 2024.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. Textworld: A learning environment for text-based games. *Computer Games Workshop at IJCAI*, 2018.

Saman Forouzandeh, Wei Peng, Parham Moradi, Xinghuo Yu, and Mahdi Jalili. Learning hierarchical procedural memory for LLM agents through bayesian selection and contrastive refinement. In *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2026.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. In *ICLR*, 2024.

Yanjiang Guo, Yen-Jen Wang, Lihan Zha, Zheyuan Jiang, and Jianyu Chen. DoReMi: Grounding language model by detecting and recovering from plan-execution misalignment. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *NeurIPS*, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *ICLR*, 2020.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *ICLR*, 2024.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Proceedings of the 6th Conference on Robot Learning (CoRL)*, 2023.

Ankush Kadu and Aswanth Krishnan. Reflexgrad: Within-episode failure recovery in LLM agents via progress-gated dual-process routing. *arXiv preprint arXiv:2511.14584*, 2025.

Tomoyuki Kagaya, Thong Jing Yuan, Yuxuan Lou, Jayashree Karlekar, Sugiri Pranata, Akira Kinose, Koki Oguri, Felix Wick, and Yang You. Rap: Retrieval-augmented planning with contextual memory for multimodal llm agents. *arXiv preprint arXiv:2402.03610*, 2024.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can llms actually correct their own mistakes? a critical survey of self-correction of llms. *Transactions of the Association for Computational Linguistics*, 12:1417–1440, 2024.

Jinyeon Kim, Cheolhong Min, Byeonghwi Kim, and Jonghyun Choi. Pre-emptive action revision by environmental feedback for embodied instruction following agents. In *Proceedings of the 8th Conference on Robot Learning (CoRL)*, 2025.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *ICLR*, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. In *NeurIPS*, 2023.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. In *NeurIPS*, 2022.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *UIST*, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, et al. ProgPrompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of llm agents. In *ACL*, 2024.

Shreyas Sundara Raman, Vanya Cohen, Ifrah Idrees, Eric Rosen, Ray Mooney, Stefanie Tellex, and David Paulius. CAPE: Corrective actions from precondition errors using large language models. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), 2024a.

Renxi Wang, Haonan Li, Xudong Han, Yixuan Zhang, and Timothy Baldwin. Learning from failure: Integrating negative examples when fine-tuning large language models as agents. *arXiv preprint arXiv:2402.11651*, 2024b.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? In *EMNLP*, 2022.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024c.

An Yang, Baosong Yang, Beichen Zhang, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Wei Yang, Jinwei Xiao, Hongming Zhang, Qingyang Zhang, Yanna Wang, and Bo Xu. Coarse-to-fine grounded memory for LLM agent planning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025b.

Zeyuan Yang, Peng Li, and Yang Liu. Failures pave the way: Enhancing large language models through tuning-free rule accumulation. In *EMNLP*, 2023.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In *NeurIPS*, 2022a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022b.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023a. URL `https://arxiv.org/abs/2305.10601`.

Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. Editing large language models: Problems, methods, and opportunities. In *EMNLP*, 2023b.

Yunxiang Zhang, Yiheng Li, Ali Payani, and Lu Wang. Adamem: Test-time adaptive memory for language agents. *arXiv preprint arXiv:2606.05684*, 2026.

Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents. *arXiv preprint arXiv:2404.13501*, 2024.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. *AAAI*, 2024.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *ICLR*, 2024.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

A ALFWorld Implementation and Statistical Audit

A.1 Separating the Two Predictor Implementations

ALFWorld uses a $20 \to 32 \to 32 \to 1$ MLP (1761 parameters), while synthetic crafting uses $11 \to 32 \to 32 \to 1$ (1473 parameters in the supplied code). The hidden widths are shared; feature maps, datasets, memory lifetimes, and checkpoints are separate. Synthetic data come from a scripted policy and expose explicit threshold failures. ALFWorld data come from bare-Qwen training-split rollouts. Its target is whether the immediate feedback is exactly `Nothing happens.`; recorded rule predictions provide a shadow reference. The target is an operational no-effect label and can miss failures expressed otherwise. It does not predict whole-episode success.

Table 3: ALFWorld features, using zero-based indices. Predicates use $1/0$ and, where applicable, -1 for unknown or not applicable.

Index	Definition
0–11	One-hot template: go to, open, close, take, put, heat, cool, clean, use, examine, look/inventory, invalid.
12	Action matches a supported command template.
13	Current location matches the target receptacle.
14	Target occurs in the initially observed receptacle set.
15	Observed open/closed status of the target.
16	Agent holds any object.
17	Held object matches the action's object.
18	Object occurs in the target's observed contents.
19	Go-to target matches the current location.

A.2 Observation-derived ALFWorld Features

Supported surface forms are canonicalized before featurization and execution. The tracker initializes receptacle names from the visible initial observation and updates location, inventory, known contents, and open/closed status from executed-action feedback. State resets at each game; the MLP stays frozen. No simulator admissible-action list enters the feature extractor.

The parser supplies task semantics. Target presence is a membership test on observed receptacles. Unknown and not-applicable values share an encoding; incomplete observations can make inferred predicates wrong. Entity names are excluded from the vector, while action-template identities remain.

A.3 Data, Optimization, and Selection

The recovered training file contains 60 records from 60 distinct task identities and 2054 steps. The collector uses the training split. With seed 0 and a 0.2 held-out fraction, the trainer shuffles record indices into 48 training and 12 validation records. A direct check of this recovered file found zero task identity overlap between the subsets. The training identities are also disjoint from the 134-task evaluation manifest. The code splits by record index, so this property must be rechecked if a replacement dataset contains repeated tasks.

The trainer performs 300 full-batch Adam updates at learning rate 10^{-3}. Training uses positive-class-weighted binary cross-entropy; the positive-class weight is computed from the effective weighted class counts. In the event variant, if canonical action string a occurs $n_{g,a}$ times in record g, each row receives weight $1/n_{g,a}$. Its repeats therefore contribute total weight one within that record, even when they occur at different states. This groups exact canonical action strings, not templates or adjacent loops. The row-weighted version assigns every row weight one.

The selected checkpoint minimizes validation BCE. Validation BCE uses the same row/event sample weighting, without the training positive-class multiplier. After restoring this checkpoint, the script selects a threshold on the *training* subset: maximize weighted F1 over $0.05, 0.10, \ldots, 0.95$ subject to weighted precision at least 0.90. If no candidate qualifies, the threshold remains 0.5. This is distinct from validation-based epoch selection and does not establish probability calibration. The recovered event checkpoint has threshold 0.5; the recovered ordinary checkpoint has threshold 0.3.

Provenance. The training-record SHA-256 is `85f6d4fd0a2abcae9e1490cc2399bb61b17c1a86af5dc3b0d7498b74c34d5abc`. The event checkpoint SHA-256 is `a1d53e3f3098bfce90a443509ecb9bdf1e80c62203a7c0e83135ea110c6e2bc6`. Machine-readable counts and hashes accompany the source.

Adaptive Development. The stored development scripts analyze false positives on evaluation trajectories and motivate event weighting. Although loss optimization uses disjoint training tasks, these diagnostics make the initial evaluation adaptive to development. The later frozen control uses the same benchmark and does not create a fresh untouched test set. We treat the reported comparisons

as exploratory and do not interpret the listed-family corrections as accounting for all development choices.

A.4 Feature Attribution and Diagnostic Language

For a veto, let $\mathcal{J}(x)$ be the engineered eligible predicate edits, and let $x^{(j)}$ change only coordinate j to a designated satisfied value. The historical learned diagnostic selects the largest positive score reduction:

$$j^* = \arg \max_{j \in \mathcal{J}(x)} \{p_\theta(x) - p_\theta(x^{(j)})\}.$$

Edits concern grammar (12), location (13), target presence (14), occupancy of hands (16), held-object match (17), observed object presence (18), and repeated navigation (19), subject to action-specific eligibility. Open/closed state (15) is an input but has no corresponding attribution edit. There is no joint-edit search. If no eligible edit lowers the score, the system returns a generic rejection. Otherwise fixed templates produce a reason and optional repair. The procedure measures model sensitivity to engineered feature edits; explanations and repairs are not simulator-validated interventions.

The new shared-feedback control does not use this MLP attribution procedure. Both gates call the same deterministic rule reason/repair formatter. A learned veto unsupported by that formatter receives generic feedback. Hence the control isolates a change in the gate within a fixed rule-informed language interface.

A.5 Initial Arm Identity and Accounting

- `app_rule`: rule prediction with a textual reason.
- `app_generic`: the rule checker with generic rejection.
- `app_rx`, `app_rx_b150`: rules with reasons and optional repair, at 50 and 150 calls. These correspond to 97/134 and 99/134.
- `app_learned`: ordinary MLP training and feature-attributed diagnostics, 90/134.
- `app_learned_event`: event-weighted MLP with feature attribution, 94/134; this variant is used for the initial cross-actor transfer.

Every generated proposal consumes an LLM call; every executed command consumes an environment step. The historical runners expose retry/exhaustion options and contain a final-budget-call path that can bypass checking. The new control checks every proposal and skips after exhausted retries. Accordingly, nominal budget equality does not make historical and control trajectories interchangeable. The initial aggregate export contains zero-valued trajectory summaries from field mismatches; those fields are not used to infer absence of vetoes or failures.

A.6 Interface and Reflection Prompts

The actor's system prompt is, verbatim: "Interact with a household to solve a task. The following is an example. Then solve the new task the same way. Reply with exactly one action per turn, starting with '> '." The actor context is the one-shot demonstration for the task type, the memory block (lessons from previous trials, if any), and the task's initial observation; each proposal is appended as "> *action*" followed by the environment response or gate message.

The gate's messages, verbatim: a generic rejection is "[Predictor] This action would fail. Choose a different action." A state-bound rejection is "[Predictor] This action would fail: *reason*. Suggested fix: *repair*." ("Choose a different action." when no repair is available; the reason-only ablation drops the fix sentence.) Exhaustion after three re-queries injects "[Predictor] Still predicted to fail; turn skipped. Re-plan from the current state." The oracle's generic message is "[Predictor] This action would fail: it is not an admissible command in the current state. Choose a different action."

The reflection call's system prompt is, verbatim: "You are an agent that failed a household task. Diagnose why in a few sentences and state a different high-level plan for the next attempt. Be concrete about which actions were wasted and what to do instead. Do not output actions, only the diagnosis and the plan." Its user message contains prior reflections, the task's initial observation, and

Table 4: Exact paired audit of the ten reported contrasts (134 common games each). The first named condition defines wins.

First / second condition	Wins/losses	p	p_{Bonf}
Qwen: rule + reason / Bare	21/12	0.162756	1.000000
Qwen: rule + generic / Bare	14/2	0.004181	0.041809
Qwen: rule + repair 50 / Bare	28/10	0.005098	0.050976
Qwen: rule + repair 150 / Bare	30/10	0.002221	0.022214
Qwen: MLP event / Bare	26/11	0.020074	0.200739
GLM: MLP 50 / Bare	7/0	0.015625	0.156250
GLM: MLP 150 / Bare	11/0	0.000977	0.009766
Reflexion 3 / MLP event	22/13	0.175465	1.000000
Reflexion 3 / rule + repair 50	20/14	0.391528	1.000000
Reflexion 3 / rule + repair 150	20/16	0.617719	1.000000

the executed trajectory (action–response pairs, truncated to 300 characters per response). Reflection output is whitespace-normalized and capped at 700 characters. The one-shot demonstrations for the seven ALFWorld task types, the full rule and repair templates, and the ScienceWorld prompts ship verbatim in the release repository under `reference_assets/` and the runners.

A.7 Exact Paired Contrasts

For w tasks won only by the first condition and ℓ won only by the second, the two-sided exact McNemar probability is

$$p = \min\left(1, 2\sum_{k=0}^{\min(w,\ell)} \binom{w+\ell}{k} 2^{-(w+\ell)}\right).$$

For $w + \ell = 0$, $p = 1$. Table 4 enumerates the ten initial reported contrasts, with adjusted values $\min(1, 10p)$. Reflexion trial 3 versus its own Bare-equivalent trial 1 is a cumulative-retry diagnostic outside this listed family. The ordinary MLP remains descriptive because its paired counts were not included in the aggregate export; marginals do not determine a paired test. The separate 16-contrast interim control family is fully listed in Appendix F; neither correction accounts for adaptive reuse or repeated interim inspection.

B Statistical Protocol, Pre-Registration, and Determinism Controls

Tests And Intervals. Paired success contrasts use the exact two-sided McNemar test defined in Appendix A.7. Paired differences of proportions carry Newcombe (method 10) confidence intervals, which account for the correlation induced by the discordant-pair counts; we quote them wherever a non-significant difference could be misread as equivalence. The reflexion family (six primary contrasts plus the 2×2 model) is Holm-corrected within family; the oracle family and the cross-actor family use Holm over their own members. A sign-flip permutation on per-task interaction contrasts ($B = 200{,}000$) supplements the parametric interaction test.

Pre-Registration And Disclosures. The analysis structure was fixed in versioned protocol files before the corresponding results were inspected: a closed factorial family over gate $\times$ feedback content (reported as published); the reflexion family (designed after those arms were run; disclosed as a post-hoc designation); the oracle family (frozen before oracle results were inspected); the ScienceWorld protocol (frozen before any cross-suite analysis); and the same-batch hybrid rerun (approved and launched before its results were seen). Secondary families (cross-suite reflexion cells, cross-batch contrasts, per-type breakdowns, actor floors) carry no confirmatory claim. All pre-registered primary results are reported, including the pre-registered nil on ScienceWorld (Section 4.2).

Cross-Batch Determinism. Greedy decoding is a deterministic function of the prompt *and the GPU model*, and the evaluation batches span two GPU architectures (NVIDIA L20 for the factorial

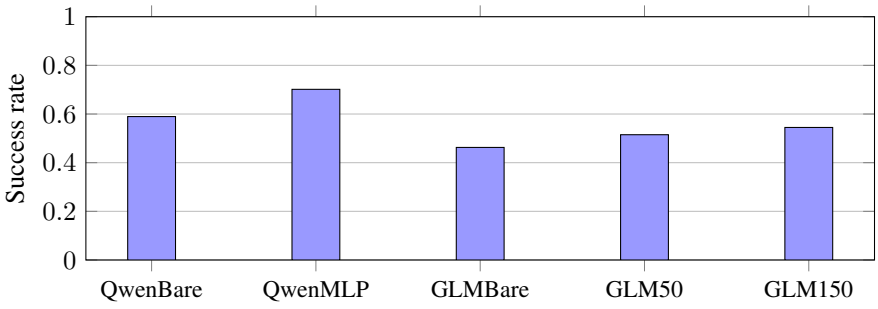

Figure 5: Cross-actor transfer of the event-weighted MLP on ALFWorld with an unchanged feature vocabulary and parser. GLM50 and GLM150 denote the two call limits.

and reflexion batches; A100-SXM4-80GB for the oracle, ScienceWorld, and rerun batches). Three checks establish that the resulting cross-batch shifts are numerics, not configuration: (i) within every batch, disjoint-cap cells of the same arm agree on all trajectory prefixes ($134/134$ games); (ii) across servers, reflexion trial 1 (L20) matches the factorial batch's bare cell on first action and outcome ($134/134$); and (iii) the first divergent generations carry identical prompt hashes yet fork at different token ids, ruling out prompt or decoding drift. Overall, $61/134$ games show at least one action flip across the two GPU models, of which 6 flip the success outcome (all in one direction, as observed and disclosed). The bare baseline therefore shifts by up to 6 of 134 games (4.5 points) between batches, while the intervened arms are far more stable (the hybrid: 119 vs. 121 of 134, eight discordant games, $p = 0.73$). All headline contrasts in the main text are within-batch and task-paired; no claim rests on a cross-batch subtraction.

Rollout Signatures. Each batch is a frozen-protocol rollout archive whose preflight signature binds source, prompts, model hashes, task manifest and per-game environment reset hashes: factorial `a58b8b14` (L20), reflexion `f31d71ad` (L20), oracle `19b557c4` (A100), ScienceWorld `ffc960f8` (A100), and the same-batch hybrid rerun `4deff0eb` (A100).

C Cross-Actor Transfer and Floor Cases

ChatGLM3-6B. The learned gate checkpoint trained on Qwen3-8B trajectories (event-weighted variant; Appendix A) deploys on ChatGLM3-6B with the feature vocabulary, observation parser, thresholds, and message policy unchanged and no retraining. Bare success is $62/134$ (46.3%); the gate raises it to $69/134$ (51.5%) at the 50-call cap and $73/134$ (54.5%) at the 150-call cap. The cap-150 contrast against bare is $+11/-0$ games (exact McNemar $p = 0.00098$; Holm-adjusted $p = 0.0098$ within its family). Figure 5 visualizes these cells.

Llama-3.2-3B-Instruct: A Floor Case. On Llama-3.2-3B-Instruct the bare baseline collapses to $1/110$ completed episodes at the 50-call cap (rule gate: $6/110$). A harness audit confirms this is a capability floor rather than a protocol artifact: proposals parse to valid actions, the observed no-effect rate is 70.6%, and episodes end on environment termination rather than budget exhaustion. The model sits below the range where interface effects are interpretable, so we exclude it from cross-actor claims and report it as a floor artifact.

D ScienceWorld Protocol

ScienceWorld (Wang et al., 2022) contributes the cross-suite test of Section 4.2. The suite was frozen before any arm ran: 25 task types $\times$ variations $\{0, 1, 2\} = 75$ tasks. Five types (three inclined-plane, two Mendelian-genetics variants) were excluded *pre-run* because the simulator's gold-path generator yields no complete demonstration for any of their variations (final score 0), making success undefinable. Four arms run at the 50-call cap (bare, rule gate, reflection memory, AFR), one evaluation batch, preflight signature `ffc960f8`.

Table 5: ScienceWorld results (batch C, 75 tasks). Comparisons are against Bare. The primary test is rules versus Bare.

Method	Success (%)	Δ (pp)	Wins/losses	p
Bare	8/75 (10.7)	—	—	—
Rules gate	9/75 (12.0)	+1.3	2/1	1.0
Reflection memory	18/75 (24.0)	+13.3	10/0	.0020
AFR	17/75 (22.7)	+12.0	10/1	.012

Failure Label. The suite's verbalized no-effect responses ("No known action matches that input." ∪ "Nothing happens.") define the failure label; informative precondition failures are *not* labeled. The label therefore defines no gate-training target on this suite (the gate runs rules-only) and claims ride on success and interception rates rather than label rates. Because the model here continues the demonstration's "> action / observation" pattern, the response parser takes the first non-empty line and strips the > marker (the ALFWorld parser needs neither adjustment).

Scores. ScienceWorld partial-credit scores can be negative; we report mean final score alongside success. Under matched budgets the reflection arm moves the mean from -15.1 (bare) to -3.7, and success from $8/75$ to $18/75$.

E Mechanism Diagnostics

All quantities in this section are computed post hoc from the frozen rollout archives (analysis only; no new runs). Trial boundaries are the `llm_call` resets; a "veto" is an intercepted proposal, and "repair lands" is the fraction of vetoes whose first executed re-proposal avoids the no-effect response.

Table 6: Veto accounting by batch. Counts are per-game means; exhausted loops reach the local re-query limit. First-repair landing is the percentage of vetoes whose first executed re-proposal avoids a no-effect response.

	Batch A (L20)		Batch B (A100)		
	rules@150	AFR	rules@150	oracle@150	AFR
Vetoes / game	6.7	7.0	8.6	11.6	6.4
First repair lands (%)	80.1	78.8	80.6	90.7	79.1
Veto loops exhausted / game	0.56	0.54	1.16	1.72	0.58

Oracle Message Composition. The oracle shares the rule gate's message policy: a veto whose action trips a state rule receives the same reason-plus-repair message, and any other veto receives a generic non-admissibility message. Composition of the oracle's vetoes: at cap 50, 72.5% specific vs. 27.5% generic (697 vetoes); at cap 150, 82.6% specific vs. 17.4% generic ($1{,}554$ vetoes). The rule gate's vetoes are 100% specific by construction, so the oracle's marginal interceptions over the rule gate are exactly its generic-message slice; the oracle comparison in Section 4.2 therefore combines re-query volume with this bounded repair reach.

Prompt-Token Attribution. In every arm the actor prompt is fixed by construction (task description plus prior actions and observations; the admissible-command list is gate-external and never enters it). Per-game prompt-token totals are nonetheless almost a linear function of veto count (Pearson $r = 0.96$ for the batch-B rule gate, 0.98 for the oracle; fitted per-veto cost 3.6k and 3.9k tokens, intercepts 24.4k vs. 22.2k at zero vetoes): each re-query re-sends the accumulated context, so a gate that vetoes more often pays proportionally more. This, rather than any per-step list context, explains the oracle's $+21\%$ prompt-token total over the rule gate.

Among the 28 games where both arms fail trial 1, common-set rescue rates are 32.1% $(9/28)$ for AFR and 17.9% $(5/28)$ for memory alone. This task-matched comparison remains descriptive. Reflection cards, written after failed trials, average 0.66 cards per episode and 85 words per card for memory alone, versus 0.44 cards and 77 words for AFR.

Table 7: Per-trial mechanics in batch A. Costs are means; success is a percentage and n counts games entering each trial. Later trials condition on earlier failure, so cross-arm rates describe different task sets. Trial 1 is bare for memory and gated for AFR.

		Memory (reflection)			AFR (gate + memory)		
		t1	t2	t3	t1	t2	t3
means	calls	29.3	40.3	45.8	26.6	36.4	42.1
	prompt tok	35.1k	56.9k	70.4k	32.9k	51.8k	64.9k
	env steps	29.3	40.3	45.8	22.7	29.5	34.6
	trial succ.	61.2	30.8	11.1	71.6	44.7	28.6
	n	134	52	36	134	38	21

F Matched-feedback Control: Frozen Protocol and Interim Audit

This appendix documents the frozen protocol of the matched-feedback control and its interim (even-ordinal, 67-task) audit. The completed 134-task factorial built on this protocol is analyzed in the main text (Section 4.2); the interim numbers below are retained for provenance and carry no confirmatory claim.

F.1 Protocol and Provenance

This control was created after the initial evaluations to separate gate identity from feedback formatting. Its manifest fixes 134 unique `valid_unseen` task paths and 60 training task paths. The two identity sets are disjoint. Sorting the evaluation paths and assigning zero-based ordinals defines two equal parity partitions. Results reported here cover even ordinals only; every one of its 67 tasks has five arms under two call caps.

Table 8: Frozen control settings. "Shared" means the same deterministic rule-based reason/repair formatter under either gate.

Item	Setting
Actor	Qwen3-8B, BF16, greedy decoding, thinking disabled
Generation	32 new tokens maximum; eager attention; seed 42 reset each episode
Tasks	134 unique `valid_unseen` paths; ordinal parity partitions
Arms	Bare; rule/generic; rule/shared; MLP/generic; MLP/shared
Budgets	50 or 150 LLM calls; 50 executed environment actions
Veto loop	At most three re-queries; skip after exhaustion; every proposal gated
Feedback	Generic and exhaustion strings byte-identical across gates; shared formatter and generic fallback identical across gates
MLP	Recovered event-weighted checkpoint; threshold 0.5; frozen weights
Actor input exclusions	No admissible-command list, expert plan, reward, evaluator success flag, predictor score, or rule verdict
Prompt supervision	One demonstration chosen by task type
Preflight	Model, tasks, reset behavior, prompts, manifest, source, and weights hashed; reset probe passed before execution

The preflight signature is `a58b8b14ccab32f9bd1fb9d584a7c69978866 2e299d876d83fa845d0aff92a53`. The event-checkpoint hash appears in Appendix A. Each episode records task identity, arm, budgets, seed, signature, outcome, termination, call/environment counts, and a trajectory. Each proposal record includes the raw generation, token IDs, token counts, truncation flag, parsed action, 20 features, rule decision/reason, MLP score/decision, intervention feedback, execution outcome, and exhaustion status. The manuscript source bundle includes the analysis JSON; full trajectories are distributed in the separate raw-results archive.

The actor sees task-type-selected demonstrations. This is explicit task-type supervision, even though expert plans and admissible commands are withheld. Shared feedback contains engineered rule semantics; the experiment controls that information between gates rather than comparing either gate

Table 9: Exploratory contrasts on the 67-task partition. Intervals are unadjusted paired-task bootstrap intervals (percentage points); $p_{16} = \min(1, 16p)$ does not account for repeated interim inspection.

Cap	Method	Comparator	Gain/loss	Δ (pp)	95% interval	p	p_{16}
50	Rule/generic	Bare	6/2	+6.0	[-1.5, 14.9]	0.2891	1.0000
50	Rule/shared	Bare	14/4	+14.9	[3.0, 26.9]	0.0309	0.4941
50	MLP/generic	Bare	6/2	+6.0	[-1.5, 14.9]	0.2891	1.0000
50	MLP/shared	Bare	13/4	+13.4	[1.5, 25.4]	0.0490	0.7847
50	MLP/generic	Rule/generic	0/0	+0.0	[0.0, 0.0]	1.0000	1.0000
50	MLP/shared	Rule/shared	1/2	-1.5	[-7.5, 3.0]	1.0000	1.0000
50	Rule/shared	Rule/generic	9/3	+9.0	[-1.5, 19.4]	0.1460	1.0000
50	MLP/shared	MLP/generic	9/4	+7.5	[-3.0, 17.9]	0.2668	1.0000
150	Rule/generic	Bare	6/2	+6.0	[-1.5, 14.9]	0.2891	1.0000
150	Rule/shared	Bare	16/4	+17.9	[6.0, 29.9]	0.0118	0.1891
150	MLP/generic	Bare	6/2	+6.0	[-1.5, 14.9]	0.2891	1.0000
150	MLP/shared	Bare	14/4	+14.9	[3.0, 26.9]	0.0309	0.4941
150	MLP/generic	Rule/generic	0/0	+0.0	[0.0, 0.0]	1.0000	1.0000
150	MLP/shared	Rule/shared	1/3	-3.0	[-9.0, 3.0]	0.6250	1.0000
150	Rule/shared	Rule/generic	11/3	+11.9	[1.5, 22.4]	0.0574	0.9180
150	MLP/shared	MLP/generic	10/4	+9.0	[-1.5, 19.4]	0.1796	1.0000

with a knowledge-free language channel. Once gates produce different decisions, their trajectories can diverge, so identical formatter code does not imply identical realized text.

F.2 Integrity and Summary

All 670 expected JSON files are present and parseable. There are 670 unique (task, arm, call-cap) keys, one preflight signature, 67 unique tasks, and ten conditions per task. Every trajectory length equals its recorded LLM-call count. No historical episode or unfinished odd-ordinal task is included in the interim audit; the completed run's aggregate cells appear in Table 1 of the main text.

Bare has identical outcomes and trajectories across the nominal 50/150 caps because its environment terminates within 50 calls in this partition; the repeated row is a protocol check, not a second independent baseline sample. Rule/generic and MLP/generic both yield 45/67 at each cap and have identical task-level success vectors. Their environment counts differ because the gate decisions differ. This illustrates why matched outcomes neither imply identical intervention nor establish gate equivalence.

F.3 Paired Analysis

The analysis defines eight directional comparisons per cap: each intervention versus Bare; MLP versus rules within each feedback mode; and shared versus generic within each gate. For a method/comparator pair, "gain" means method succeeds and comparator fails on the same task; "loss" is the reverse. Exact two-sided McNemar tests use the definition in Appendix A.7. A single Bonferroni family multiplies each unadjusted value by 16. Paired task bootstrap intervals sample the 67 paired task rows with replacement 10,000 times, using seed 20260909. Intervals are empirical 2.5th and 97.5th percentiles and are unadjusted. They do not capture model-selection uncertainty, actor-seed variability, or repeated interim inspection.

No adjusted comparison crosses 0.05. The smallest unadjusted value is for rule/shared versus Bare at 150 calls: 16 gains, 4 losses, $p = 0.011818$, adjusted $p = 0.189087$. The corresponding observed difference is 17.9 percentage points and its unadjusted paired bootstrap interval is [6.0, 29.9] points. At 50 calls, the same contrast is 14 gains and 4 losses, $p = 0.030884$, adjusted $p = 0.494141$. MLP/shared versus Bare has 13/4 at 50 calls and 14/4 at 150 calls. These are positive sample trends with negative task-level cases and no corrected confirmatory conclusion.

For MLP/generic versus rule/generic, all 67 observed task-level differences are zero at both caps. The empirical task bootstrap consequently returns $[0, 0]$. That degenerate resampling result reflects this fixed sample; it is not evidence of population equivalence. An equivalence margin, design, and test were not specified.

Table 10: Initial ALFWorld `valid_unseen` results for frozen Qwen3-8B, separating rule and learned interfaces.

Interface	Success	Rate	Calls	Env	$+/-$	p
Bare	79/134	0.590	29.8	29.8	–	–
Rule + reason	88/134	0.657	27.6	23.4	21/12	0.162756
Rule + generic	91/134	0.679	27.0	22.2	14/2	$0.004181^{\dagger}$
Rule + repair ($b = 50$)	97/134	0.724	26.5	22.2	28/10	0.005098
Rule + repair ($b = 150$)	99/134	0.739	34.8	24.5	30/10	$0.002221^{\dagger}$
MLP	90/134	0.672	27.0	21.1	–	–
MLP (event-weighted)	94/134	0.701	26.9	22.4	26/11	0.020074

$+/-$: games gained/lost against Bare. Calls and Env are per-game means. †: passes the 0.05 threshold after Bonferroni correction within the listed ten-contrast family (exploratory). Displayed p values are unadjusted; – denotes no reported paired test.

F.4 Cost and Termination

At 50 calls, rule/shared lowers mean calls from 28.60 to 25.28, environment steps from 28.60 to 21.49, and total tokens from 35,408 to 31,713 relative to Bare. MLP/shared uses 26.16 calls, 22.31 environment steps, and 33,447 tokens. Under the 150-call cap, rule/shared and MLP/shared average 48,415 and 52,694 tokens. More permissive call budgets therefore provide small observed success increments at substantially higher token volume.

`environment_done` is an environment termination category, not a success label. In this partition all 67 Bare runs terminate through the environment, while only 41 succeed. At 50 calls, call-budget termination occurs for 22/16/22/17 runs in rule/generic, rule/shared, MLP/generic, and MLP/shared. At 150 calls it occurs 4/1/4/1 times. Timing is recorded but omitted from the main comparison because shared hardware load can affect wall time.

F.5 Output-quality Audit Requirement

A generation is labeled truncated when it reaches the 32-token cap and its final token is not EOS. Under Bare at 50 calls, the mean is 24.78 truncated generations out of 28.60 calls (86.6%). Other arms also show high counts. Inspection finds examples in which a complete first-line action is followed by model-generated text resembling an observation. The runner parses an action from the output; therefore truncation does not by itself imply an invalid proposal.

A dedicated audit should distinguish: whether the first action is complete, which substring the parser retains, whether trailing generated text enters the next context, and how these outcomes relate to interventions and success. Changing token limits or stopping rules inside the unfinished partition would violate the frozen comparison. Any such variant requires a separate protocol and matched reruns, including Bare.

F.6 Completion Rule

The odd-ordinal partition was subsequently completed under the unchanged manifest, checkpoint, prompts, source hashes, and analysis code, with all ten cells retained per task; the combined 134-task results are those of Section 4.2 in the main text. Task removal, threshold adjustment, selective stopping, or mixing historical cells would have invalidated the control, and none occurred.

G Initial (Pre-Control) ALFWorld Runs

Before the matched-feedback control of Appendix F froze the protocol, an initial set of interface variants was evaluated on the same 134 tasks (Table 10). These runs informed the design of the shared-feedback control and are retained for provenance; the completed factorial in the main text supersedes them for all claims. Figure 6 visualizes them, and Figure 7 shows the observed success–

cost operating points of the initial reflexion port; budgets are not matched in these early runs and no claim rides on them.

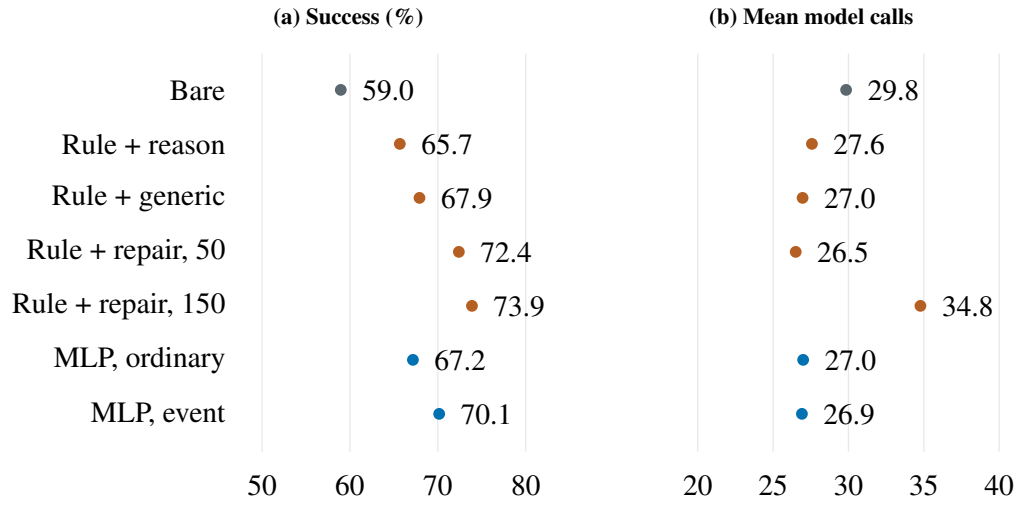

Figure 6: Success and call costs for the initial Qwen3-8B interface comparisons. Colors identify interface families; values correspond to Table 10 and are descriptive.

H LIMITATIONS

Each cell is a single deterministic run. Decoding is greedy and deterministic per GPU model, but not across GPU models: the bare baseline shifts by 6 of 134 games between two evaluation batches (A100 vs. L20 hardware numerics; identical prompts, forked argmax on near-ties), while intervened arms shift by only 8 discordant games, a 2-game net difference. All headline contrasts are therefore within-batch and task-paired; the cross-batch hardware spread is reported as a measurement caveat rather than a finding (Appendix B).

The engineered components encode ALFWorld-specific knowledge. The rule predicates, 20-dim featurizer, and repair templates were derived from this suite; the cross-suite nil result is the honest measure of that specificity, and the tracker would need re-derivation for a new environment class. The learned gate is tiny by design (it tests the interface role of prediction, not the frontier of failure prediction); much larger detectors could behave differently, though the oracle bounds what *perfect* detection buys under this budget structure.

The statistics also have known scope, and the single-run design is the main limitation of this study. Every cell is one deterministic run: uncertainty is over tasks (paired tests), not sampling seeds or decoding temperature, so run-to-run variance is unmeasured while the cross-GPU numeric spread shows that near-tie outcomes are sensitive; multi-seed or sampled-decoding replications are the natural next step. the ScienceWorld suite (75 tasks) is powered for medium-to-large effects only, and its cells sit in the pre-registered secondary family. Several family designations are post-hoc (the reflexion family was designed after its arms were observed); we disclose each where it is used. The learned gate's development loop inspected early evaluation trajectories and motivated the event weighting (Appendix A); the shared-feedback control constrains, but does not eliminate, this adaptive reuse; family corrections do not absorb benchmark reuse or interface iteration. The synthetic-environment studies of the previous manuscript version are retained as illustrative appendix material only (Appendix I).

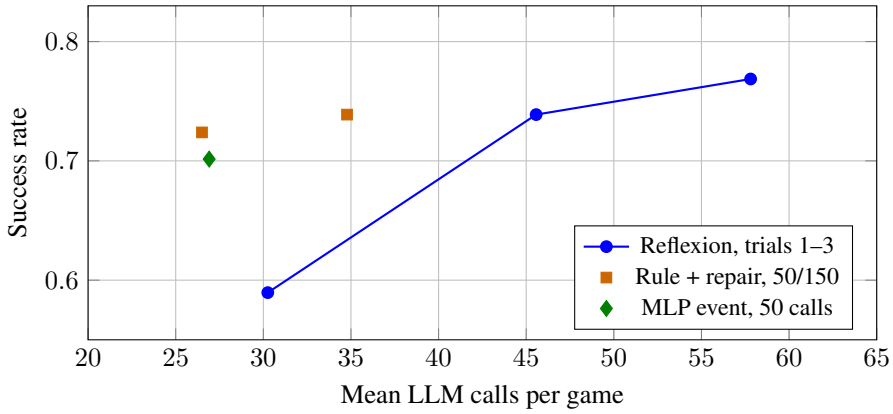

Figure 7: Descriptive success–cost comparisons: the Reflexion line connects cumulative trials, while rule and MLP markers represent separate single-episode conditions. Budgets are not matched.

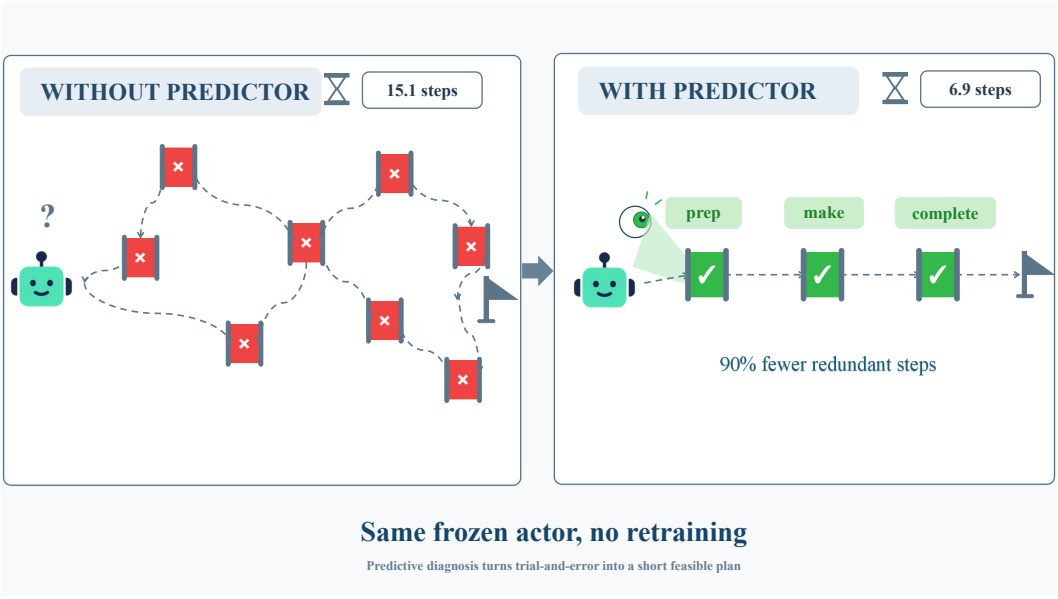

Figure 8: Illustrative trajectories in the synthetic crafting environment, with and without learned predictive diagnosis on held-out family C. The actor is frozen in both conditions; aggregate results appear in Table 11.

I Synthetic Environment: Task and Interface Comparisons (Previous Manuscript Version)

This appendix retains, for completeness, the synthetic-environment studies of the previous manuscript version. They illustrate the interface mechanisms in a setting where preconditions and optimal step counts are explicit, enabling descriptive comparisons. Because the interfaces differ in information content and memory persists across episodes, these results do not isolate independent causal effects of formatting, state binding, and repair. **Verifiability disclaimer:** the historical synthetic checkpoints and run artifacts are absent from the current snapshot, so these results cannot currently be reconstructed or independently verified from the release; the appendix is retained as provenance of the manuscript's development, not as verifiable evidence. Figure 8 illustrates the interface's effect on that world.

Table 11: Synthetic results with a frozen actor: mean capped steps over ten sequential episodes, with success counts in parentheses. APP trains on the other three families, excluding the held-out family's key names and API identifiers.

Held-out task	Arm mean steps (success/10)					*Optimal*
	Bare LLM	**Text lessons** (LS)	**Structured** table (ST)	**Predictor** (rule, PR)	**Predictor** (learned, APP)	
family A	4.3 (10)	11.5 (10)	4.4 (10)	3.6 (10)	**3.3** (10)	3
family B	10.0 (10)	7.9 (10)	5.3 (10)	**4.1** (10)	4.6 (10)	4
family C	15.1 (10)	22.7 (6)	11.6 (10)	**6.5** (10)	6.9 (10)	6
family D	13.4 (10)	17.6 (6)	8.5 (10)	**5.5** (10)	**5.5** (10)	5

I.1 World and Failure Signal

We use a synthetic crafting world (a weaving task). A task family defines a set of entities, tools, and preconditions; the agent must reach a target product count. Characteristically, a `prep` action sets a readiness key (e.g. `ready_c`) that a `make` action consumes, and *the tool is reset after every* `make`, so a fresh `prep` is required before each `make`. This reset is what makes a naive actor waste steps: it re-calls `make` right after a success. Each action declares conditions such as `Need ready_c >= 1`; when unsatisfied, the environment returns "Precondition not met: ..." and the state is unchanged. This message is the only teaching signal about conditions.

Task Families. Families A–D are in-distribution for training prediction checkpoints. Two out-of-distribution *structural* families are added: E is a sequential double `prep` (one `prep`'s precondition is another `prep`'s output); F has an *exhaustible* source (`gather` itself has a precondition, `src_f >= 1`, and the resource does not renew, so the task margin tolerates only one wasted call). In the training families `gather` always succeeds, so F is a genuine out-of-distribution test of the "no infeasible advice" factor.

Auto-Finish. The models we evaluate do not reliably call `complete_task` when done. To keep termination behavior out of the measured quantity we strip it: achieving the goal auto-completes the episode. This removes a non-studied noise source and lets us measure the number of *actions* taken.

Actors and Measurements. The primary actor is `Qwen2.5-3B-Instruct` Yang et al. (2024); a second, stronger actor `Qwen3-8B` Yang et al. (2025a) tests transfer across model scale. We report mean steps to completion over 10 seeded episodes (memory carries across episodes; seed 0 is a cold start) and the number of precondition errors. Failed episodes are counted as 25 steps (the episode cap). This defines a capped-cost metric, not uncensored time to completion; it does not uniformly favor either arm.

I.2 Main Five-arm Result

I.3 Descriptive Effect Sizes and Dependence

Table 12 summarizes the observed paired step differences. The 40 episode pairs belong to four persistent-memory sequences: episode t can change the knowledge available at episode $t+1$. Episode-wise sign flips and bootstrap resampling would ignore this dependence, so we do not interpret the previous episode-level p values or intervals as valid uncertainty estimates for independent learning runs. Future repetitions should reset memory between independent sequences and resample at the sequence level.

The earlier Monte Carlo description also used 10^5 random sign flips, which cannot support a corrected Monte Carlo estimate below $1/(10^5+1)$ when using $(k+1)/(B+1)$. We therefore retain the descriptive effects rather than claim unsupported $p < 10^{-6}$ precision. This does not change the observed step counts.

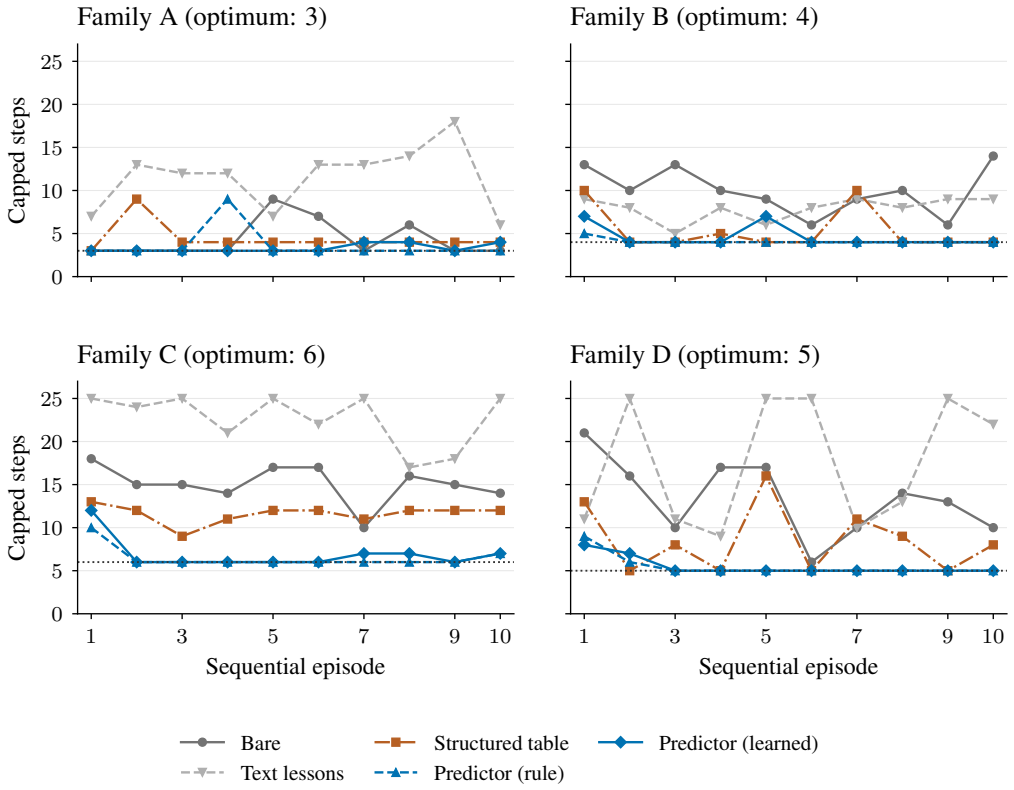

Figure 9: Capped steps across sequential episodes with persistent memory for held-out task families. Dotted lines mark family-specific optima; the trajectories are descriptive and do not establish equivalence between predictors.

Table 12: Descriptive paired step differences from Bare across four ten-episode sequences. Negative values mean fewer steps; failures count as 25 steps. No independence-based intervals or significance tests are reported.

Arm	Mean paired Δ steps
Text lessons (LS)	+4.22
Structured table (ST)	−3.25
Predictor (rule, PR)	−5.78
Predictor (learned, APP)	−5.62

I.4 Joint Changes to Coverage and Repair

LESSON and STATIC differ in both representation and task-specific knowledge, so their ordering cannot be assigned solely to prose versus structure. PRED adds current-state evaluation and actionable repair. PRED-PA additionally lists all actions and suppresses advice to replenish a non-renewable source. On family F, conventional PRED has two capped episodes and a mean of 9.8 steps; PRED-PA has no capped episodes and a mean of 6.7. This comparison changes coverage and repair policy together, rather than isolating either factor.

I.5 Out-of-Distribution Structural Families

Table 13 reports E and F, where the structure itself is outside the training distribution. On E (sequential double-prep) all predictor variants reach near-optimal behavior (4.5 vs. bare 9.0; optimum 4). On F (exhaustible source) the conventional rule format fails as described, and both PRED-PA and APP

Table 13: OOD families: E, sequential double-prep; F, an exhaustible source. Entries are mean capped steps; parentheses give deadlock counts. APP reportedly trained on A/B/D, but its weights remain unrecovered. Rule variants check remembered preconditions without neural checkpoints.

Task	Bare	PR (rule, conv. format)	PR-PA (rule, per-action)	APP
E	9.0	4.5	4.5	**4.5**
F	8.8	9.8 (2 deadlock)	6.7 (0)	**6.2** (0)
optimal				E 4 / F 6

Table 14: Qwen3-8B: mean capped steps over ten sequential episodes, using the reported APP checkpoint `pred_mlp_nb_C.pt`. Bold marks the lowest arm mean; no independent sequence-level uncertainty estimates are available.

Task	Bare	PR-PA	APP	optimal
C	8.0	7.8	**6.1**	6
F	8.3	**6.2**	6.3	6

succeed (6.7 and 6.2, optimal 6, zero deadlocks). This illustrates a combined coverage-and-repair effect under an OOD structure, without isolating the contribution of either change.

I.6 A Second, Stronger Actor

Table 14 repeats C and F with Qwen3-8B. On C, the reported excess over the six-step optimum decreases from 2.0 to 0.1 steps and precondition errors from 20 to 1. Bare repeatedly proposes `make` after it has reset the tool. This behavior is consistent with a failure to apply the readiness condition at the current state; it does not reveal what the actor internally knows. The observations concern one persistent-memory sequence per condition.

J Synthetic Training Data Generation

A scripted *random agent* acts on the training families, producing $\sim$480 runs ($\sim$20K steps, 34% failure rate) of `pred_data.jsonl`. Two variants: the baseline generator, and an anti-shortcut variant (`--no-retry-bias`) that removes the "retry a failed API" bias, lowering the failure rate to 13%. The main results use the baseline; the anti-shortcut variant is used for the robustness analysis in Appendix M. No LLM is used in data generation.

K Synthetic Predictor Architecture and Features

The supplied synthetic implementation constructs eleven anonymous features: number of known gates, any unsatisfied gate, worst threshold margin, all gates satisfied, action success count, action failure count, never-tried indicator, known resource increase, known key-setting effect, steps since reset, and `gate_key_moved`. Raw key names and API identifiers are excluded. `PredMLP` implements $11 \to 32 \to 32 \to 1$ (1473 parameters), with ReLU hidden layers and a sigmoid score at inference. Its training entry point uses BCE, Adam at 10^{-3}, batches of 256, and a default of 30 epochs. It saves the last epoch; it has no separate validation-based checkpoint selection. These are the audited code settings. The historical synthetic checkpoint files are absent from this snapshot, so their tensor shapes and exact run arguments remain unverified.

At evaluation, threshold and effect tables are populated from visible failure messages and state differences, and persist across episodes. Training excludes the held-out family. The reported rule-support count is $126/126$: error text such as `Need X >= n` exposes thresholds directly. This strong supervision channel differs from ALFWorld's operational no-effect label and limits claims about discovering unobserved task semantics.

L CONDITIONAL FEATURE ABLATION

The supplied feature-ablation scripts examine threshold-related coordinates, including `gate_key_moved`, on examples with known preconditions. Such an analysis should distinguish discrimination on the complete dataset from behavior on this more informative subset. Zeroing features can also create out-of-distribution combinations; a drop in score discrimination alone would not exclude other shortcuts.

The historical AUC values are withheld in this revision. The supplied `auc` helper sorts (score, label) pairs and assigns consecutive ranks without averaging ties. For example, two examples with equal scores and labels 0 and 1 receive AUC 1.0 under this helper, whereas the tie-correct value is 0.5. Historical predictions and checkpoint provenance must be recovered and evaluated with a standard tie-aware implementation before the AUC claims are restored. This issue does not alter the reported online step counts.

M ROBUSTNESS: ANTI-SHORTCUT RETRAINING

A generator variant removes the bias toward retrying failed actions. The original records report a failure-rate change from 34% to 13%. Its online means are A 3.2, B 4.2, C 6.9, and D 5.7 steps, with 40/40 success, compared with baseline means 3.3/4.6/6.9/5.5. These are descriptive outcomes of the tested generator change. Historical AUC comparisons require the tie-aware recomputation described above; sequence-level repeats are needed for inference about online robustness.

N OOD FAMILIES AND SECOND ACTOR: PER-SEED DATA

N.1 E/F PER-SEED STEPS

E: bare 9.0; rule PRED 4.5; PRED-PA 4.5; APP 4.5 (optimal 4). F: bare 8.8; rule PRED 9.8 (2 deadlocks, lengths $[25, 6, 25, 6, \dots]$); PRED-PA 6.7 (0); APP 6.2 (0; optimal 6).

N.2 SECOND ACTOR PER-SEED STEPS

Qwen3-8B, C: bare 8.0 ($[8, \dots]$ constant); PRED-PA 7.8; APP 6.1 (optimal 6). F: bare 8.3; PRED-PA 6.2; APP 6.3 (optimal 6).

N.3 PRED-PA VS APP DIVERGENCE ON C (8B)

On C, PRED-PA is nearly ineffective for 8B (7.8, 18 errors) while APP works (6.1, 1 error); on F both work. The only textual difference is APP's probability annotation. Reported as an observation; mechanism unknown.

O THE EXACT PROMPTS, ARM BY ARM

All five arms share the same skeleton: observation, an arm-specific injected block, the last six outcomes, and a format example. The following is a real replayed step from family C (episode/seed 1, step 2): the state immediately after a successful `make`, i.e. the exact moment of the bare actor's blind spot: the tool has been reset (`ready_c=0`), which is visible in the state but requires connecting that state to the next action's precondition.

Shared Skeleton (The BARE Arm Is Exactly This).

```
Task: Produce 3 units of product R, then call complete_task().
You have enough inv_c already; extra gathering does not help.
Current state: inv_c=5, prod_r=1, ready_c=0
Available APIs: op_clean_tool_c
op_gather_inv_c
op_make_prod_r
```

```
op_noop_c
op_prep_tool_c
Goal progress: prod_r = 1 / needs >= 3
  (call complete_task() when met)

<ARM-SPECIFIC BLOCK INSERTED HERE>

Recent outcomes:
  op_prep_tool_c() -> [NONE] (ok)
  op_make_prod_r() -> [NONE] (ok)

The lines below are FORMAT examples only - the names are
fictional placeholders, not real APIs:
some_fn()
some_fn(idx=1)

Next action:
```

LESSON Block (Free Text, From Other Families).

```
Lessons learned in previous environments (different API names,
similar structure):
- Making actions have hidden preconditions. An error
  'Precondition not met: Need X >= n' states exactly what is
  missing; satisfy it and the same call will succeed.
- If a RESOURCE is insufficient, call the action that increases
  that resource, then retry.
- Some making actions also need a READINESS key set to 1 by a
  prep-like action first.
- Preconditions must hold at the moment of each call: making may
  consume resources or reset keys, so re-check before every
  attempt.
- Do not repeat an action that succeeds but does not move the
  goal counter.
```

The narrative mentions that making may reset keys. Its generic content differs from the task-specific memory supplied by the structured conditions.

STATIC Block (Current-Family Memory, Structured, Not State-Checked).

```
Known preconditions (learned from earlier failures in this
environment):
  op_make_prod_r (this action moves the goal counter):
    requires ready_c >= 1
Known effects:
  op_make_prod_r: sets ready_c to 0
  op_make_prod_r: sets inv_c to 5
  op_make_prod_r: increases prod_r
  op_prep_tool_c: increases ready_c
```

PRED Block (Rule, State-Bound, Conventional Format).

```
Precondition check (learned from earlier failures in this
environment):
  op_make_prod_r (this action moves the goal counter):
    - ready_c = 0 -> NOT MET (need >= 1)
      - call op_prep_tool_c (it increases ready_c)
```

APP Block (Learned Judgment, State-Bound, Per-Action).

```
Action outcome prediction (learned model, based on failures
seen so far):
  op_clean_tool_c: predicted ok (p=0.13)
  op_gather_inv_c: predicted ok (p=0.13)
```

```
  op_make_prod_r (this action moves the goal counter):
    predicted FAIL (p=1.00) - ready_c=0 needs >=1
    (call op_prep_tool_c)
  op_noop_c: predicted ok (p=0.13)
  op_prep_tool_c: predicted ok (p=0.00)
```

The rule-based per-action arm (PRED-PA) prints the same per-action list without the probabilities; that probability annotation is the only textual difference behind the 8B divergence reported in Appendix N.3. All blocks above are generated verbatim by replaying the recorded main-table trajectory through the same code path that produced the results.

P Qwen3-8B Selection Correction

An early pass eliminated Qwen3-8B as unusable ($0/10$ all-PARSE). Root cause: a `thinking`-mode flag (`enable_thinking=False`) was passed wrapped inside `chat_template_kwargs`, which the Qwen3 template does not recognize, so thinking was never disabled and `<think>` output consumed the entire token budget. Fixing the argument (passing `enable_thinking=False` directly to `apply_chat_template`) restored the actor ($3/3$, 8.0 steps). This is a methodological caution: a single parameter-passing error can fully invert a model-selection conclusion.

Q Comparison with Reported Results of Related Systems

Table 15 collects additional ALFWorld literature results. The same-backbone comparison is in Table 1. Each row reports what the original paper reports, under its own actor, task split, demonstration allowance, and call budget; the entries are therefore *not directly comparable across rows*. We keep the table to published numbers rather than reimplementations: rerunning these systems under our protocol would change their demonstration and budget conditions and would no longer be the systems their authors evaluated. The one row sharing our actor and task count (ReflexGrad, Qwen3-8B, 134 tasks) differs in demonstration allowance and decoding configuration, which its authors note shifts the bare baseline itself (35.1% zero-shot at n=10 seeds versus our 61.2% single-seed with one-shot demonstrations).

Our contribution is orthogonal to these rows: each system couples its external knowledge into one evaluated pipeline, whereas AFR holds the actor and budget fixed and measures the now channel (gate), the later channel (memory), their combination, and an admissibility-oracle comparison, with task-paired tests inside each batch. The comparison rows provide methodological context. Differences in models, splits, and budgets preclude a cross-paper performance ranking.

Table 15: Reported ALFWorld results of related frozen-actor systems. All numbers are quoted from the cited papers under their own protocols and are not directly comparable across rows (different actors, splits, demonstrations, and budgets). "Offline build" marks systems whose external knowledge is constructed before deployment. AFR numbers are our two ALFWorld evaluation batches (Table 1, blocks A and B).

Method	Venue	Actor	Demos / build	ALFWorld	Offline
AutoManual (Chen et al., 2024)	NeurIPS'24	GPT-4-turbo	1 demo	97.4	rules
CFGM (Yang et al., 2025b)	EMNLP'25	GPT-4-Turbo	offline library	91.0	memory
MACLA (Forouzandeh et al., 2026)	AAMAS'26	Llama-2-7B	offline library	90.3 (unseen)	memory
AdaMEM (Zhang et al., 2026)	arXiv	Qwen3-4B	offline library	58.2 (unseen)	memory
ReflexGrad (Kadu & Krishnan, 2025)	arXiv	Qwen3-8B	none	75.4	none
AFR (ours)	–	Qwen3-8B	1 demo	88.8 / 90.3	none

CFGM uses GPT-4-Turbo as its actor and GPT-4o for knowledge extraction, with cross-validation and a 20-step ALFWorld limit. MACLA uses frozen, 4-bit Llama-2-7B. The AdaMEM entry denotes the training-free AdaMEM-low variant with Qwen3-4B-Instruct-2507 and on-policy training-

split memory. These auxiliary resources and protocols differ from AFR and are not controlled comparisons.