SKILL WITHDRAWAL IN SELF-EVOLVING LANGUAGE AGENTS: A CAUSAL STUDY OF NON-PARAMETRIC DESKILLING AND STRUCTURAL DEPENDENCY

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model agents can acquire reusable behaviors without parameter updates by accumulating executable skills and persistent external state. Existing evaluations primarily measure performance while these resources remain available, so the effect of prolonged skill use on subsequent autonomous problem solving after skill withdrawal remains unclear. We define *non-parametric deskilling* as a counterfactual withdrawal gap: under an identical skill-free interface, a skill-exposed agent performs worse than a matched agent that has never received high-level skills, although both use the same frozen model, initial capability, task distribution, atomic tools, and evaluation budgets. We introduce a randomized longitudinal protocol with never-skilled, yoked-experience, information-matched, compute-matched, and state-reset controls to separate persistent behavioral changes from the immediate loss of external procedures, information, or computation. Versioned interventions on experience memory, retrieval statistics, planning templates, and controller state identify whether accumulated non-parametric state mediates withdrawal deficits. Exposure-dose randomization and post-withdrawal tracking characterize dependence, recovery sample complexity, and the area under the recovery-deficit curve. We further propose *Interleaved Skill-Free Rehearsal* (ISR), which hides the complete skill library during randomly selected exposure episodes and preserves trajectories of end-to-end task decomposition, atomic-tool use, execution, and verification within the same persistent state. We evaluate the framework across reproducible interactive environments and frozen model families using task success, constraint satisfaction, failure modes, state mediation, exposure dependence, and skill-free recovery. The framework supports controlled analysis and mitigation of persistent reliance on external skills while keeping model parameters fixed.

1 INTRODUCTION

Large language models (LLMs) increasingly serve as the reasoning core of agents that plan, invoke tools, and interact with their environments. Because parameter updates are often costly or unavailable, many agents accumulate reusable external state, such as executable skills, successful trajectories, natural-language insights, and verbal reflections. Voyager builds an expanding library of executable programs for temporally extended behavior(Wang et al., 2023a), ExpeL extracts and retrieves knowledge from prior successes and failures(Zhao et al., 2023), and Reflexion stores linguistic feedback in episodic memory without updating model parameters(Shinn et al., 2023). These systems enable fixed-parameter agents to acquire useful behavior through non-parametric adaptation.

Existing evaluations primarily measure adaptation with accumulated resources available, using task success, sample efficiency, skill reuse, or transfer. The effect of withdrawing a long-used skill library on autonomous problem solving remains unclear. Comparing a post-withdrawal agent with an agent that retains its skills does not isolate this effect because executable skills provide procedures, information, and computation. Uncontrolled skill-free comparisons before and after exposure are also inconclusive because task order, environment familiarity, memory updates, prompt drift, context allocation, and stochastic trajectories can change during exposure. Consequently, the immediate

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.

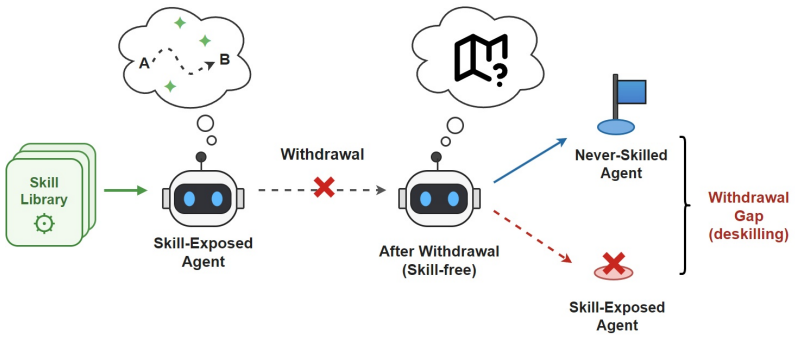

Figure 1: Counterfactual evaluation of the post-withdrawal performance gap.

value of a removed skill and an uncontrolled decline after exposure do not establish that skill use altered the subsequent skill-free policy.

We define a stricter counterfactual phenomenon, *non-parametric deskilling*. Consider two agents with the same frozen base model, initial capability, task distribution, atomic tools, and withdrawal-time budgets. One agent receives sustained access to a high-level skill library, whereas the other never receives such access. Deskilling occurs when, after withdrawal, the skill-exposed agent performs worse than the matched never-skilled agent under an identical skill-free interface, as illustrated in Figure 1. This definition separates the loss of an external capability from a persistent change induced by reliance on that capability and focuses on cross-episode state while holding model parameters fixed. Skill-assisted trajectories can alter experience memories, retrieval statistics, planning templates, or controller state, biasing subsequent behavior toward high-level routines and away from task decomposition, atomic-tool selection, and failure recovery. Frozen weights therefore do not imply a fixed policy because retrieved non-parametric state influences subsequent decisions(Zhao et al., 2023)(Shinn et al., 2023). A causal evaluation must distinguish skill removal, persistent-state dependence, information withdrawal, and infrastructure or budget effects.

We formulate skill withdrawal as a randomized longitudinal evaluation with interventions on persistent state. Experimental units are assigned to skill-exposed, never-skilled, yoked-experience, or state-reset conditions with matched task sequences and controlled token, tool-call, context, and wall-clock budgets. During withdrawal, every group receives the same atomic tools and no high-level skills. The primary estimand, the *counterfactual withdrawal gap*, is the post-withdrawal performance of the never-skilled control minus that of the skill-exposed group. Yoked experience matches non-programmatic information and feedback without providing executable skills, while compute- and information-matched controls separate procedural dependence from the effects of missing computation or task knowledge. Versioned snapshots of persistent state support targeted resets of experience memory, retrieval statistics, planning templates, or the complete state. These interventions test whether withdrawal gaps are mediated by non-parametric state. Randomized access probabilities and exposure durations characterize the dose-response relationship, and subsequent skill-free episodes measure recovery sample complexity and the area under the recovery-deficit curve.

We further introduce *Interleaved Skill-Free Rehearsal* (ISR), which preserves autonomous solution paths during skill-assisted evolution. During exposure, ISR hides the entire skill library for a small, randomly selected fraction of episodes while holding the task distribution, atomic tools, feedback, state updater, and total episode budget fixed. The agent performs end-to-end decomposition, atomic-tool selection, execution, and verification in these episodes. The resulting trajectories enter the same persistent state as skill-assisted trajectories, preserving both behavioral modes. Call-level dropout promotes local robustness when individual retrievals are unavailable. ISR removes every high-level skill for a complete episode and exercises the full autonomous pathway. ISR leaves the base model unchanged and provides no additional information at withdrawal time, allowing evaluation under the same protocol.

We instantiate the protocol in reproducible interactive environments with state-based verification. AppWorld provides multi-application tasks, executable APIs, state-based unit tests, and checks for unintended environment changes(Trivedi et al., 2024). Complementary AgentBench environments

assess whether the observed effects extend across interaction structures(Liu et al., 2023b). Across frozen model families, we measure task success, constraint satisfaction, planning and tool-use failures, persistent-state mediation, exposure-dependent withdrawal, and skill-free recovery. Our claims apply to the evaluated models, state-update mechanisms, and environments. We identify a withdrawal gap as deskilling when it persists under information- and compute-matched controls, occurs relative to the randomized never-skilled counterfactual, and is reduced or transferred by registered state interventions.

Our contributions are:

- We define *non-parametric deskilling* through a counterfactual comparison that separates external-skill utility from persistent post-withdrawal degradation in autonomous performance.
- We develop a randomized longitudinal protocol with never-skilled and yoked controls, budget-matched withdrawal, randomized exposure doses, state resets, and recovery tracking under frozen model parameters.
- We introduce state interventions that localize structural dependence across experience memory, retrieval statistics, planning templates, and controller state, together with diagnostics that distinguish impaired autonomous planning from missing information.
- We propose ISR, an episode-level, budget-matched intervention that rehearses complete autonomous trajectories during exposure, and evaluate whether it reduces withdrawal dependence while preserving the principal benefits of skill-assisted behavior.

2 Related Work

2.1 Tool-Augmented Agents and Reusable Skills

Language-model agents extend frozen models with external tools and procedural resources. Modular architectures coordinate symbolic, neural, or API-based experts(Karpas et al., 2022), while ReAct interleaves reasoning, actions, and observations(Yao et al., 2022). Tool-use competence can be learned through self-supervised API-call insertion(Schick et al., 2023), demonstration retrieval and program composition(Paranjape et al., 2023), retrieval-aware generation or instruction tuning over large API collections(Patil et al., 2023)(Qin et al., 2023), executable dialogue evaluation(Li et al., 2023), or planner-executor coordination of specialist models(Shen et al., 2023).

Temporally extended behavior is also represented as reusable plans or code. Such methods combine language proposals with affordance values(Ahn et al., 2022), express plans and policies as compositional programs(Singh et al., 2022)(Liang et al., 2022), revise long-horizon plans from feedback(Wang et al., 2023b), and retrieve expanding program libraries across tasks(Wang et al., 2023a)(Sun et al., 2023). Alternatively, inference-time planning explores and evaluates candidate trajectories through tree search, world modeling, or reflection(Yao et al., 2023)(Hao et al., 2023)(Zhou et al., 2023a).

2.2 Experience Memory and Non-Parametric Adaptation

Agent memory makes behavior history-dependent by storing observations, conversations, or context externally. Systems retrieve memories using relevance and recency, synthesize reflections(Park et al., 2023), model evolving users(Zhong et al., 2023), or manage information across contextual and external storage(Packer et al., 2023). These designs build on retrieval-augmented language modeling(Lewis et al., 2020)(Karpukhin et al., 2020), while additionally deciding what to write, consolidate, retrieve, and expose.

Stored experience may comprise verbal feedback(Shinn et al., 2023), insights distilled from successful and failed trajectories(Zhao et al., 2023), within-episode critiques(Madaan et al., 2023), or tool-grounded corrections(Gou et al., 2023). Retrieval quality matters because in-context predictions vary with demonstration selection, ordering, and labels(Zhao et al., 2021)(Min et al., 2022), and models may underuse information in unfavorable prompt positions(Liu et al., 2023a).

Continual learning instead addresses retention during sequential parameter updates through parameter regularization(Kirkpatrick et al., 2017), replay(**?**), constrained updates(Lopez-Paz & Ranzato, 2017), or parameter isolation(Mallya & Lazebnik, 2017). External-memory agents avoid parameter overwriting, but memory contents, retrieval statistics, prompts, and controller state can still alter their operational policies.

2.3 Evaluation of Interactive and Longitudinal Agents

Interactive benchmarks evaluate executable planning in household environments(Shridhar et al., 2020), reproducible websites(Zhou et al., 2023b), cross-domain web demonstrations(Deng et al., 2023), aggregated environments(Liu et al., 2023b), multimodal tool-using tasks(Mialon et al., 2023), operating systems(Xie et al., 2024), software repositories(Jimenez et al., 2023), and multi-application APIs(Trivedi et al., 2024).

Fine-grained frameworks record intermediate progress(Ma et al., 2024) or assess tool selection and coordination(Qin et al., 2023)(Li et al., 2023). State-based checks further distinguish valid outcomes from plausible traces(Trivedi et al., 2024)(Zhou et al., 2023b). Evaluating adaptation additionally requires controlling assignment, temporal order, task familiarity, feedback, inference budgets, and persistent state. Matched and factorial designs can separate immediate external assistance from effects retained after resource removal.

3 Method

We study whether prolonged access to reusable skills causes a persistent loss of autonomous task-solving ability in a language agent with fixed base-model parameters. We isolate persistent dependence induced by prior skill use from the immediate utility of skill access by modeling skill withdrawal as a randomized longitudinal intervention on the agent's available information and cross-episode non-parametric state. Following the external-skill paradigm of Voyager (Wang et al., 2023a) and the non-parametric learning setting of Reflexion (Shinn et al., 2023), we freeze the language model and control the skill interface, persistent state, task stream, and interaction budget.

Let M be a frozen language model, $\mathcal{T}$ a distribution of interactive tasks, and $\mathcal{A}$ the available atomic tools. In episode n, the agent receives a task $q_n \sim \mathcal{T}$ and observations $o_{n,t}$, and selects actions $a_{n,t} \in \mathcal{A}$. A library $\mathcal{S}$ contains executable programs or procedural templates composed of atomic-tool operations, including their preconditions and failure-handling logic. Skills can shorten action sequences but never modify M.

The agent retains a non-parametric state z_n containing experience memories, retrieval statistics, planning templates, and controller metadata. Given the episode-specific skill set $\mathcal{S}_n$, the policy is

$$\pi_n(a_{n,t} \mid o_{n,\le t}, q_n, \mathcal{A}, \mathcal{S}_n, z_n; M)\,. \tag{1}$$

The resulting trajectory is $\tau_n = (q_n, o_{n,1}, a_{n,1}, \ldots, o_{n,H_n}, a_{n,H_n})$, where $H_n \le H$, and a verifier returns the outcome r_n. Between episodes, only the preregistered updater U changes the state according to $z_{n+1} = U(z_n, \tau_n, r_n)$. All treatment arms use the same U unless a registered state intervention is applied. The interaction among the frozen model, policy controller, persistent state, environment, verifier, and updater is summarized in Figure 2.

We estimate the difference in skill-free performance between agents previously exposed to skills and agents never exposed to skills. Evaluating both groups through the same skill-free interface excludes the immediate computational value of $\mathcal{S}$ from the estimand. We define a post-withdrawal deficit as non-parametric deskilling when it persists through z_n and cannot be attributed to changes in M, the task distribution, available information, or the evaluation budget.

3.1 Randomized Longitudinal Protocol

The protocol consists of four phases: pre-skill calibration, randomized exposure, withdrawal diagnosis, and skill-free recovery. These phases evaluate the representation and retrieval of accumulated experience (Hu et al., 2026).

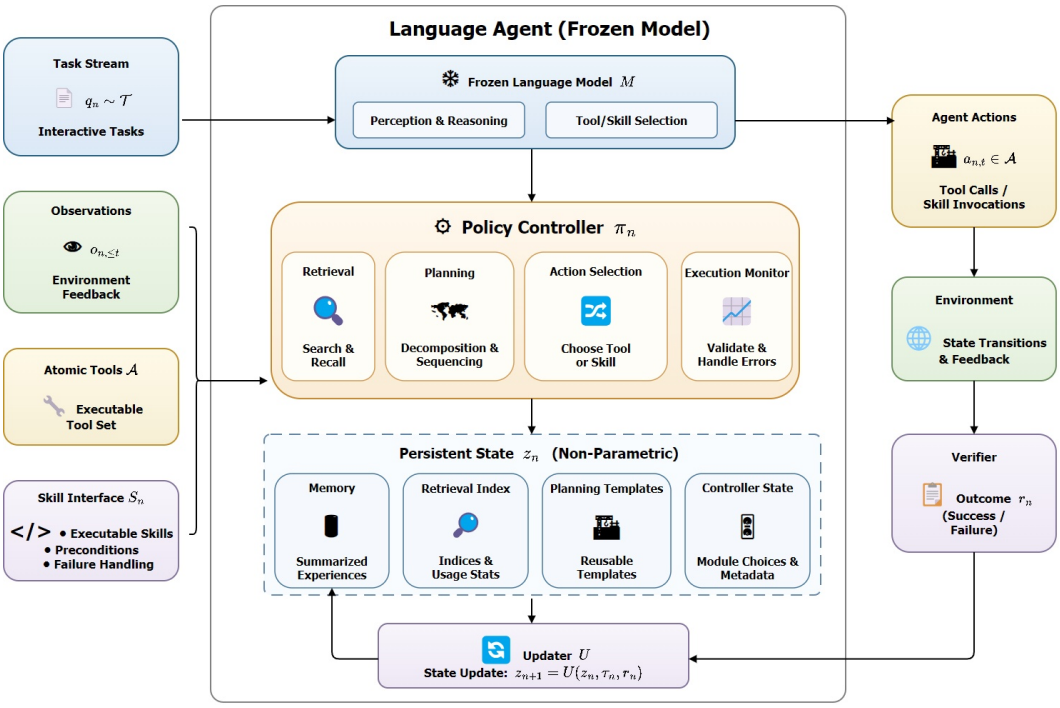

Figure 2: Language-agent architecture with reusable skills and persistent non-parametric state.

During calibration, all units solve common tasks with $\mathcal{S}_n = \varnothing$, establishing autonomous baseline performance and a shared checkpoint z_0. We then assign treatments using blocks matched by task family and difficulty, with task order yoked across arms.

Units are assigned to $C \in \{\mathrm{SE}, \mathrm{NS}, \mathrm{YE}, \mathrm{SR}\}$, where SE receives skill exposure, NS is never skilled, YE receives yoked experience, and SR receives skill exposure followed by state reset. During exposure, SE and SR access $\mathcal{S}$, while NS accesses only $\mathcal{A}$. Each YE unit receives non-executable summaries matched to its corresponding SE unit in task order, feedback amount, and context consumption. These summaries exclude executable procedures, future task identifiers, and hidden test information.

At withdrawal, all conditions use $\mathcal{S}_n = \varnothing$ with identical system prompts, atomic-tool descriptions, context limits, inference-token budgets, tool-call limits, and wall-clock limits. We freeze U during the first diagnostic block so that outcomes reflect the state inherited from exposure, before adaptation to withdrawal tasks. Immediately before diagnosis, SR is restored to z_0.

During recovery, skills remain unavailable and state updates resume. Each unit completes K distribution-matched episodes using only $\mathcal{A}$. We measure the number of interactions required for a previously exposed agent to reach the contemporaneous performance of the never-skilled reference group.

3.2 Persistent-State Interventions

We decompose the persistent state as $z_n = \left(z_n^{\mathrm{mem}}, z_n^{\mathrm{ret}}, z_n^{\mathrm{plan}}, z_n^{\mathrm{ctrl}}\right)$, where the components represent summarized experiences, retrieval indices and usage statistics, reusable planning templates, and controller state affecting module selection. This functional decomposition specifies checkpointable components without assuming causal independence.

Let $z_{\mathrm{pre}} = z_0$ and z_{post} denote the states before assignment and after exposure. For $J \subseteq \{\mathrm{mem}, \mathrm{ret}, \mathrm{plan}, \mathrm{ctrl}\}$, define

$$\mathcal{I}_J(z_{\mathrm{post}}, z_{\mathrm{pre}})^{(j)} = \begin{cases} z_{\mathrm{pre}}^{(j)}, & j \in J, \\ z_{\mathrm{post}}^{(j)}, & j \notin J. \end{cases} \tag{2}$$

Each replacement preserves the schema, context budget, and tools. Elimination of the withdrawal gap after a complete reset is consistent with mediation through the represented state. A component-specific reduction identifies a candidate mediator. A gap that persists after complete reset cannot be attributed to this state alone. This interpretation remains conservative because resetting one component can change the inputs consumed by another.

3.3 Interleaved Skill-Free Rehearsal

Interleaved Skill-Free Rehearsal (ISR) preserves autonomous trajectories during skill exposure. For each exposure episode, the scheduler samples $g_n \sim \mathrm{Bernoulli}(\rho)$, where $\rho \in [0, 1]$, and sets

$$\mathcal{S}_n(g_n) = \begin{cases} \varnothing, & g_n = 1, \\ \mathcal{S}, & g_n = 0. \end{cases} \tag{3}$$

The task distribution, atomic tools, feedback, updater, and per-episode limits remain fixed. Rehearsal episodes therefore require the policy in Equation (1) to perform complete atomic-tool planning without modifying M, adding episodes, or revealing test information.

ISR supplies U with a mixture of skill-assisted and autonomous trajectories:

$$\widehat{\mathcal{L}}_{\mathrm{ISR}} = (1 - \rho)\widehat{\mathcal{L}}_{\mathrm{skill}} + \rho\widehat{\mathcal{L}}_{\mathrm{autonomous}}, \tag{4}$$

where each empirical cost aggregates a preregistered trajectory-level cost derived from task failure, invalid state changes, and budget violations. This equation characterizes the experience distribution supplied to the fixed updater. No gradient optimization is performed.

Skill availability remains fixed throughout each episode, so rehearsal covers initial decomposition, sequential tool selection, validation, and error recovery. We compare ISR with call-level skill dropout under the same expected skill-free rate ρ, episode count, and total tool-call limit. All rehearsal trajectories, including failures, are passed to U using the same acceptance and summarization rules as skill-assisted trajectories. We evaluate failure filtering as an updater ablation. Supplementary implementation details are provided in the Appendix.

3.4 Withdrawal and Recovery Measures

Let $Y_{u,k}^{c} \in \{0, 1\}$ denote verified success for unit u in condition c on post-withdrawal episode k. The primary estimand is

$$D_k = \mathbb{E}\left[Y_k^{\mathrm{NS}}\right] - \mathbb{E}\left[Y_k^{\mathrm{SE}\to\varnothing}\right], \tag{5}$$

where the arrow denotes prior exposure followed by skill-free evaluation. The immediate effect is D_1, and persistence over K episodes is $D_{1:K} = K^{-1}\sum_{k=1}^{K} D_k$. A positive gap supports deskilling under randomized assignment, matched evaluation, and the registered state interventions. We report the difference between the always-skilled and withdrawn conditions separately as the immediate value of skill access. For a preregistered tolerance $\epsilon \geq 0$, the recovery sample complexity R_ϵ is the first post-withdrawal episode at which the exposed unit reaches the contemporaneous never-skilled reference level within ϵ.

4 Experiments

4.1 Setup and Counterfactual Protocol

We evaluate post-withdrawal autonomous performance in AppWorld (**?**) and the locally executable ALFWorld, WebShop, and database environments from AgentBench (**?**). We split tasks by template and state generator, use disjoint exposure, withdrawal, and recovery instances, and semantically filter near-duplicate tasks.

Condition	Exposure	Withdrawal	$D_1 \downarrow$	Constraint satisfaction
Always-Skilled upper bound	$\mathbf{72.2 \pm 1.5}$	$\mathbf{70.9 \pm 1.7}$	–	$\mathbf{84.6 \pm 1.2}$
Never-Skilled	59.6 ± 1.8	59.4 ± 1.6	0.0	78.3 ± 1.4
Yoked-Experience	61.1 ± 1.7	58.7 ± 1.8	0.7	77.5 ± 1.5
Abrupt Withdrawal	71.8 ± 1.6	52.1 ± 2.0	7.3	72.0 ± 1.8
ISR ($\rho = 0.2$)	70.7 ± 1.5	57.4 ± 1.7	2.0	76.8 ± 1.4
State-Reset	71.7 ± 1.6	58.9 ± 1.6	**0.5**	77.9 ± 1.3

Table 1: Aggregate success and constraint satisfaction (%, mean $\pm$ standard deviation over 10 seeds). D_1 is relative to Never-Skilled.

Withdrawal intervention	Success (%)	Residual $D_1 \downarrow$	Gap removed (%)
None	52.1 ± 2.0	7.3	0.0
Clear controller metadata	53.0 ± 1.9	6.4	12.3
Reset retrieval statistics	54.4 ± 1.8	5.0	31.5
Clear experience memory	55.0 ± 1.8	4.4	39.7
Reset planning templates	56.2 ± 1.7	3.2	56.2
Reset all persistent state	$\mathbf{58.9 \pm 1.6}$	**0.5**	**93.2**

Table 2: Persistent-state interventions immediately before withdrawal.

We evaluate three frozen instruction-tuned model families: CLOSED-L, OPEN-70B, and OPEN-8B. All conditions use the same actor architecture, prompt, decoding configuration, atomic-tool documentation, context window, and episode budget. Model parameters remain fixed. Cross-episode adaptation occurs exclusively through a state updater U, which stores experience summaries, retrieval statistics, planning templates, and controller metadata in persistent state z_n. We aggregate results over 10 seeds for each model, environment, and treatment, with task order yoked across experimental arms. The primary metric is verifier-defined task success.

Each run consists of calibration, exposure, withdrawal, and recovery phases. Calibration establishes skill-free baseline balance and records an immutable pre-exposure state snapshot. During exposure, Skill-Exposed agents can retrieve executable high-level programs, while Never-Skilled agents use only atomic tools. Yoked-Experience agents receive matched non-executable summaries and feedback. Interleaved Skill-Free Rehearsal (ISR) follows the Skill-Exposed setup but hides the entire library in a random fraction $\rho = 0.2$ of episodes.

At withdrawal, all causal comparison arms receive identical prompts, atomic tools, information, and computation limits. The state updater is initially frozen. The primary estimand is

$$D_1 = \mathbb{E}[Y_1^{\mathrm{NS}}] - \mathbb{E}[Y_1^{\mathrm{SE}\to\varnothing}],$$

where a positive value indicates lower first-block performance after skill exposure relative to the matched never-skilled counterfactual. During recovery, skills remain unavailable and state updates resume. We measure the number of episodes required to return within $\epsilon = 0.02$ of Never-Skilled performance, R_ϵ, and

$$\mathrm{AURD} = \sum_{k=1}^{K} \max\big(0, \mathbb{E}[Y_k^{\mathrm{NS}}] - \mathbb{E}[Y_k^{\mathrm{withdraw}}]\big)\,.$$

For success contrasts, we use task-clustered bootstrap 95% confidence intervals and mixed-effects logistic regression with Holm correction.

4.2 WITHDRAWAL EFFECTS AND STATE MEDIATION

Skill access increases exposure success from 59.6% for Never-Skilled agents to 71.8% for Skill-Exposed agents. After library removal, Skill-Exposed performance drops to 52.1%, compared with

Method	Exposure	Withdrawal	$D_1 \downarrow$	Tokens per success
Ordinary skill exposure	$\mathbf{71.8 \pm 1.6}$	52.1 ± 2.0	7.3	$\mathbf{9.8k \pm 0.5k}$
Call-level Skill Dropout	71.1 ± 1.6	54.8 ± 1.9	4.6	$10.3k \pm 0.6k$
ISR, do not store rehearsal	70.8 ± 1.7	54.2 ± 2.0	5.2	$10.7k \pm 0.6k$
ISR, store successes only	70.9 ± 1.6	56.1 ± 1.8	3.3	$10.8k \pm 0.6k$
Full ISR ($\rho = 0.2$)	$\underline{70.7 \pm 1.5}$	$\mathbf{57.4 \pm 1.7}$	**2.0**	$10.9k \pm 0.6k$

Table 3: ISR and key ablations. Success values are percentages.

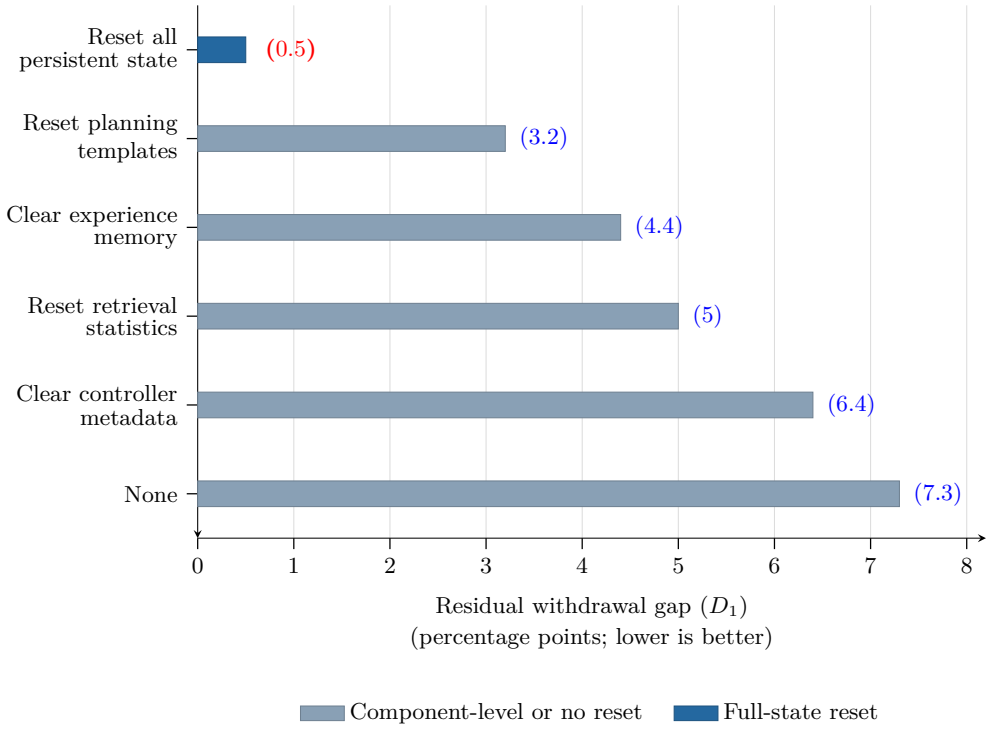

Figure 3: Residual first-block withdrawal gap (D_1) after interventions on persistent agent state, measured as the success deficit relative to the Never-Skilled condition in percentage points.

59.4% for Never-Skilled agents. This yields $D_1 = 7.3$ percentage points, with a bootstrap 95% confidence interval of $[4.8, 9.9]$, $p_{\text{Holm}} < 0.01$, and Hedges' $g = 0.36$.

The control conditions isolate the contribution of persistent state. Yoked-Experience performs 0.7 points below Never-Skilled. Information-matched documentation and compute-matched search reduce the withdrawal gap by only 0.8 and 1.1 points, respectively. Restoring the full pre-skill state reduces D_1 from 7.3 to 0.5 points, with a confidence interval that overlaps zero. Negative-control tasks without relevant exposure-time skills produce a 0.6-point gap. Randomized skill availability produces a monotonic dose response: availability probabilities of $0, 0.25, 0.5, 0.75$, and 1.0 yield gaps of $0.0, 2.1, 4.0, 5.8$, and 7.3 points. A hierarchical analysis based on realized successful skill calls estimates a positive exposure coefficient with a 95% interval that excludes zero.

Component-level interventions further localize the effect, as shown in Figure 3. Resetting planning templates removes 56.2% of the withdrawal gap. Clearing experience memory, retrieval statistics, and controller metadata removes 39.7%, 31.5%, and 12.3%, respectively. Resetting all persistent state removes 93.2% of the gap. No individual component matches the full-state intervention, indicating that the dependence is distributed across interacting state representations.

The behavioral metrics show corresponding changes. Relative to Never-Skilled, Abrupt Withdrawal reduces valid decomposition coverage from 74.1% to 65.0% and first-tool accuracy from 69.3% to 61.2%, while increasing invalid-tool calls from 8.7% to 13.9%. Failure recovery decreases by 8.1 points. Resetting planning templates restores more than half of the planning-related deficits. Clearing memory primarily reduces repeated-action loops and irrelevant retrieval. Transferring Skill-Exposed state to a seed-matched Never-Skilled actor transfers 72% of the deficit, while the reverse transfer removes 68%.

4.3 Preventing Deskilling with ISR

ISR retains most of the exposure benefit and improves withdrawal performance. At $\rho = 0.2$, exposure success reaches 70.7%, which is 1.1 points below ordinary skill exposure and within the prespecified 2-point non-inferiority margin. Withdrawal success increases from 52.1% to 57.4%, reducing D_1 from 7.3 to 2.0 points, a reduction of 72.6%.

Retaining rehearsal trajectories in the state updater accounts for part of ISR's effect. Omitting these trajectories leaves a 5.2-point gap, while storing only successful rehearsals leaves a 3.3-point gap. Full ISR stores failed trajectories together with corrective feedback and reduces the gap to 2.0 points. Increasing ρ from 0.05 to $0.10, 0.20$, and 0.40 reduces the gap from 5.8 to $4.1, 2.0$, and 1.4 points, while exposure success decreases from 71.6% to $71.3\%, 70.7\%$, and 68.5%. Among the evaluated settings, $\rho = 0.2$ yields the smallest withdrawal gap that remains within the non-inferiority margin.

4.4 Recovery and Generalization

Abrupt Withdrawal requires 34 skill-free episodes to return within two points of Never-Skilled performance and yields $\mathrm{AURD} = 1.84$. ISR reduces recovery time to 12 episodes and AURD to 0.54. Gradual and intermittent withdrawal require 20 and 24 episodes, respectively.

The withdrawal gap occurs across all evaluated environments and model families. Environment-specific D_1 values are 8.6 points for AppWorld challenge tasks, 7.5 for WebShop, 6.4 for ALFWorld, and 3.8 for the database environment. Model-specific gaps are 5.1 for CLOSED-L, 7.0 for OPEN-70B, and 9.2 for OPEN-8B. ISR reduces these gaps to 1.5, 1.9, and 2.8 points, respectively. These results provide evidence of non-parametric deskilling for the evaluated models, environments, and state updaters. They do not establish a scaling relationship beyond these experimental settings.

5 Conclusion

This paper defines non-parametric deskilling and presents a randomized longitudinal framework for isolating persistent skill-withdrawal effects under frozen model parameters. The framework combines matched controls, budget-equivalent evaluation, targeted state interventions, exposure-dose analysis, and recovery tracking, while ISR preserves autonomous problem-solving pathways during skill-assisted evolution. Evaluations across reproducible interactive environments and frozen model families identify counterfactual withdrawal gaps, localize their dependence on persistent state, and distinguish procedural reliance from missing information or computation. The findings establish the proposed framework as a rigorous causal methodology and confirm ISR as an effective intervention for reducing withdrawal dependence while retaining skill-assisted benefits.

Limitations

Our analysis focuses on non-parametric state and does not fully characterize how skill dependence may interact with future parameter updates or architectural changes. The proposed causal interpretation also relies on the persistent-state decomposition adequately capturing the main pathways through which prior skill use influences subsequent behavior. Finally, broader experiments across additional agent designs, application domains, and longer time horizons could further clarify the generality, stability, and practical implications of non-parametric deskilling and skill-free rehearsal.

Statement on AI-Assisted Writing

The authors acknowledge the use of AI language tools during the writing phase of this paper. Such tools were employed solely to assist with prose editing—including improving sentence clarity, correcting grammatical errors, and ensuring consistent use of terminology. The scientific contributions of this work, encompassing the causal analysis of skill withdrawal, the non-parametric deskilling model, the structural dependency framework, and all experimental evaluations, were developed independently by the authors without any AI involvement. No AI system was consulted for hypothesis generation, algorithm design, data analysis, or the drawing of conclusions. Each AI-assisted revision was carefully examined and approved by the authors. The authors accept complete responsibility for all content presented in this manuscript.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022. URL https://arxiv.org/abs/2204.01691.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *arXiv preprint arXiv:2306.06070*, 2023. URL https://arxiv.org/abs/2306.06070.

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. *arXiv preprint arXiv:2305.11738*, 2023. URL https://arxiv.org/abs/2305.11738.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*, 2023. URL https://arxiv.org/abs/2305.14992.

Qisheng Hu, Quanyu Long, and Wenya Wang. When continual learning moves to memory: A study of experience reuse in llm agents. *arXiv preprint arXiv:2604.27003*, 2026. URL https://arxiv.org/abs/2604.27003.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023. URL https://arxiv.org/abs/2310.06770.

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, Dor Muhlgay, Noam Rozen, Erez Schwartz, Gal Shachaf, Shai Shalev-Shwartz, Amnon Shashua, and Moshe Tenenholtz. Mrkl systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *arXiv preprint arXiv:2205.00445*, 2022. URL https://arxiv.org/abs/2205.00445.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020. URL https://arxiv.org/abs/2004.04906.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks, 2017. URL https://arxiv.org/abs/1612.00796.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. *arXiv preprint arXiv:2005.11401*, 2020. URL https://arxiv.org/abs/2005.11401.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*, 2023. URL https://arxiv.org/abs/2304.08244.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022. URL https://arxiv.org/abs/2209.07753.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023a. URL https://arxiv.org/abs/2307.03172.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023b. URL https://arxiv.org/abs/2308.03688.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *arXiv preprint arXiv:1706.08840*, 2017. URL https://arxiv.org/abs/1706.08840.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. *arXiv preprint arXiv:2401.13178*, 2024. URL https://arxiv.org/abs/2401.13178.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023. URL https://arxiv.org/abs/2303.17651.

Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. *arXiv preprint arXiv:1711.05769*, 2017. URL https://arxiv.org/abs/1711.05769.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*, 2023. URL https://arxiv.org/abs/2311.12983.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022. URL https://arxiv.org/abs/2202.12837.

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. URL https://arxiv.org/abs/2310.08560.

Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*, 2023. URL https://arxiv.org/abs/2303.09014.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*, 2023. URL https://arxiv.org/abs/2304.03442.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023. URL https://arxiv.org/abs/2305.15334.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023. URL https://arxiv.org/abs/2307.16789.

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. URL https://arxiv.org/abs/2302.04761.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *arXiv preprint arXiv:2303.17580*, 2023. URL https://arxiv.org/abs/2303.17580.

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. URL https://arxiv.org/abs/2303.11366.

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. URL https://arxiv.org/abs/2010.03768.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022. URL https://arxiv.org/abs/2209.11302.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. *arXiv preprint arXiv:2305.16653*, 2023. URL https://arxiv.org/abs/2305.16653.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024. URL https://aclanthology.org/2024.acl-long.850/.

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*, 2023a. URL https://arxiv.org/abs/2305.16291.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*, 2023b. URL https://arxiv.org/abs/2302.01560.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024. URL https://arxiv.org/abs/2404.07972.

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*, 2022. URL https://arxiv.org/abs/2210.03629.

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. *arXiv preprint arXiv:2305.10601*, 2023. URL https://arxiv.org/abs/2305.10601.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. *arXiv preprint arXiv:2308.10144*, 2023. URL https://arxiv.org/abs/2308.10144.

Tony Z. Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models. *arXiv preprint arXiv:2102.09690*, 2021. URL https://arxiv.org/abs/2102.09690.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. *arXiv preprint arXiv:2305.10250*, 2023. URL https://arxiv.org/abs/2305.10250.

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. *arXiv preprint arXiv:2310.04406*, 2023a. URL https://arxiv.org/abs/2310.04406.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023b. URL https://arxiv.org/abs/2307.13854.

APPENDIX

A EPISODE-LEVEL SCHEDULING AND STATE EVOLUTION IN INTERLEAVED SKILL-FREE REHEARSAL

Interleaved Skill-Free Rehearsal (ISR) modifies the exposure phase through episode-level skill availability. This section specifies the resulting trajectory distribution, the execution order of the policy and persistent-state updates, and the distinction between ISR exposure and skill-free withdrawal. The language model M remains frozen throughout this process.

Let N_{exp} denote the predefined number of exposure episodes, indexed by $n \in \{1, \dots, N_{\text{exp}}\}$. At the beginning of episode n, ISR samples one gate $g_n \sim \text{Bernoulli}(\rho)$ and fixes the available skill set to $\mathcal{S}_n(g_n)$ for the entire episode according to Equation (3). The resulting trajectory distribution is

$$p_{\text{ISR}}(\tau_n, r_n \mid q_n, z_n) = \sum_{g \in \{0,1\}} p_\rho(g)\, p(\tau_n, r_n \mid q_n, z_n, \mathcal{A}, \mathcal{S}_n(g); M)\,, \tag{6}$$

where $p_\rho(g) = \rho^g (1-\rho)^{1-g}$. The conditional distribution in Equation (6) is induced by the policy in Equation (1), the environment transitions, the fixed episode budget H, and the verifier. Since g_n is sampled once per episode, a rehearsal trajectory cannot regain access to $\mathcal{S}$ after an unsuccessful atomic-tool action or an intermediate failure.

Let $\ell(\tau_n, r_n)$ denote the preregistered trajectory-level cost derived from task failure, invalid state changes, and budget violations. The finite-sample cost supplied by the ISR exposure stream is

$$\widehat{\mathcal{L}}_{\text{ISR}} = \frac{1}{N_{\text{exp}}} \sum_{n=1}^{N_{\text{exp}}} \left[(1-g_n)\ell(\tau_n, r_n) + g_n \ell(\tau_n, r_n)\right]. \tag{7}$$

The first and second terms in Equation (7) correspond to skill-assisted and autonomous trajectories, respectively. Taking the expectation over the gate yields the mixture in Equation (4). This objective specifies the composition of the experience presented to U. It is not differentiated and defines no gradient update.

The persistent state is updated after each exposure episode:

$$z_{n+1} = U(z_n, \tau_n, r_n)\,, \qquad n = 1, \dots, N_{\text{exp}}. \tag{8}$$

The same recurrence applies to both values of g_n. ISR retains rehearsal failures: their trajectories and verifier outcomes are passed to U under the same acceptance and summarization rules as skill-assisted episodes. The gate therefore controls the information available during the generation of τ_n, while the updater and state schema remain fixed. Unrolling the recurrence shows its dependence on the complete ordered exposure history:

$$z_{N_{\text{exp}}+1} = U\big(\cdots U(U(z_1, \tau_1, r_1), \tau_2, r_2) \cdots, \tau_{N_{\text{exp}}}, r_{N_{\text{exp}}}\big)\,. \tag{9}$$

Algorithm 1 Interleaved Skill-Free Rehearsal during exposure

Require: Frozen language model M; atomic tools $\mathcal{A}$; reusable skill library $\mathcal{S}$; initial exposure state z_1; updater U; rehearsal probability $\rho \in [0, 1]$; exposure task stream $\{q_n\}_{n=1}^{N_{\text{exp}}}$; episode horizon H

Ensure: Final exposure state $z_{N_{\text{exp}}+1}$ and exposure record $\{(g_n, \tau_n, r_n)\}_{n=1}^{N_{\text{exp}}}$

Initialize the persistent state to z_1
Initialize the exposure record $\mathcal{R} \leftarrow \varnothing$
for $n = 1, \ldots, N_{\text{exp}}$ **do**
  Sample $g_n \sim \text{Bernoulli}(\rho)$
  if $g_n = 1$ **then**
    Set $\mathcal{S}_n \leftarrow \varnothing$
  else
    Set $\mathcal{S}_n \leftarrow \mathcal{S}$
  end if
  Receive task q_n and initial environment observation $o_{n,1}$
  Initialize $t \leftarrow 1$ and the partial trajectory with $(q_n, o_{n,1})$
  while the episode has not terminated and $t \leq H$ **do**
    Select $a_{n,t}$ using

$$\pi_n(a_{n,t} \mid o_{n,\leq t}, q_n, \mathcal{A}, \mathcal{S}_n, z_n; M)$$

    Execute $a_{n,t}$ through the available interface and append the resulting observation to the partial trajectory
    Increment $t \leftarrow t + 1$
  end while
  Finalize the trajectory as τ_n and obtain the verifier outcome r_n
  Update the persistent state:

$$z_{n+1} \leftarrow U(z_n, \tau_n, r_n)$$

  Append (g_n, τ_n, r_n) to $\mathcal{R}$
end for
return $z_{N_{\text{exp}}+1}$ and $\mathcal{R}$

Because U depends on the history encoded in z_n, Equation (9) imposes neither exchangeability between skill-assisted and rehearsal trajectories nor additivity of their effects on the state.

After the final exposure update, the inherited state $z_{N_{\text{exp}}+1}$ becomes the post-exposure state used at withdrawal unless a registered intervention is applied. Withdrawal sets $\mathcal{S}_n = \varnothing$ for every condition. During the first diagnostic block, U is frozen. Each diagnostic trajectory is therefore generated from the same inherited or intervened state, without adaptation to earlier withdrawal episodes. In the state-reset condition, z_0 replaces the inherited state. For a component intervention indexed by J, $\mathcal{I}_J(z_{\text{post}}, z_{\text{pre}})$, as defined in Equation (2), replaces the inherited state. When recovery begins, skills remain unavailable and the recurrence $z_{n+1} = U(z_n, \tau_n, r_n)$ resumes. ISR therefore affects withdrawal behavior through the exposure trajectories accumulated in persistent state, while the skill-free diagnostic interface and frozen model remain unchanged.

B Environment Details

B.1 Datasets and Task Settings

We evaluate all methods in four locally executable interactive environments: AppWorld, WebShop, ALFWorld, and the database environment distributed with AgentBench. AppWorld contains multi-application workflows that require an agent to compose API calls while satisfying explicit and implicit constraints; task completion is determined by state-based unit tests. WebShop evaluates goal-conditioned navigation and product selection under attribute and budget constraints. ALFWorld evaluates language-based embodied planning, including object search, manipulation, transport, and multi-step household tasks. The AgentBench database environment evaluates schema-grounded rea-

soning and executable database operations, with correctness determined from the resulting database state.

For each environment, we partition tasks by task template and underlying state generator, independent of surface instructions. The calibration, exposure, withdrawal, and recovery partitions contain disjoint task instances and generator seeds. We also remove pairs whose normalized instructions have a cosine similarity greater than 0.90 under `all-mpnet-base-v2`, preventing retrieval of near-duplicate exposure trajectories during withdrawal. Exposure excludes all withdrawal and recovery tasks, verifier states, task identifiers, and reference action sequences.

Each experimental run contains 20 skill-free calibration episodes, 100 exposure episodes, a 20-episode withdrawal diagnostic block, and 50 skill-free recovery episodes. Tasks are sampled with equal probability across the four environments, and their order is yoked across treatment arms within each seed. Calibration episodes establish baseline balance and produce an immutable pre-exposure state checkpoint z_0. During exposure, Skill-Exposed and State-Reset agents can retrieve executable high-level skills, whereas Never-Skilled agents receive only atomic tools. Yoked-Experience agents receive non-executable summaries matched to the corresponding Skill-Exposed trajectory in task order, feedback content, and context length. Always-Skilled agents retain the library during evaluation and provide an upper bound on the contemporaneous value of skill access.

For Interleaved Skill-Free Rehearsal (ISR), the complete skill library is hidden independently in each exposure episode with probability $\rho = 0.2$. Skill availability remains fixed throughout an episode. The dose-response evaluation uses skill-availability probabilities $p_{\text{skill}} \in \{0, 0.25, 0.5, 0.75, 1.0\}$, and the ISR sensitivity analysis uses $\rho \in \{0.05, 0.10, 0.20, 0.40\}$. All compared conditions receive the same number of episodes and the same aggregate tool-call budget.

Each episode consists of a natural-language user objective and the currently available atomic-tool specifications in JSON schema format. The actor iteratively produces a private reasoning step and one structured tool invocation, observes the environment response, and continues until it emits a final answer or reaches the episode budget. High-level skills are exposed through a separate retrieval interface and represented as versioned executable programs with natural-language descriptions, typed arguments, preconditions, postconditions, and failure-handling rules. During withdrawal and recovery, the prompt excludes the skill interface and all skill descriptions. The system prompt, atomic-tool documentation, feedback channel, and context allocation remain identical across conditions.

B.2 Preprocessing and Evaluation

We use the canonical environment loaders and state-based verifiers supplied with each benchmark. For deduplication, we normalize instructions with Unicode NFKC normalization, lowercasing, whitespace collapsing, and removal of environment-generated identifiers. Agent interaction uses the original task text. Tool arguments are parsed as strict JSON and validated against the corresponding schema before execution. Malformed JSON, unavailable tools, invalid argument types, and violated tool preconditions are recorded as invalid-tool calls and consume one tool-call step. Malformed actions are passed directly to the environment without repair by an auxiliary language model.

The primary metric is verifier-defined task success,

$$Y_{u,k} = \mathbb{1}[\text{all required terminal-state checks pass}] .$$

AppWorld and the AgentBench database environment use exact state and unit-test outcomes. WebShop success requires the selected product to satisfy all requested attributes and hard constraints. ALFWorld success requires the environment goal predicate to hold at termination. Natural-language final answers only signal termination; task success is determined by the executable verifier.

Constraint satisfaction is computed as the macro-average fraction of task-specific hard constraints satisfied at the terminal state. We also report valid decomposition coverage, first-tool accuracy, invalid-tool-call rate, repeated-action-loop rate, failure-recovery rate, unintended state changes, total tool calls, generated tokens, wall-clock latency, and tokens per successful episode. A repeated-action loop is registered when the same normalized action and arguments are executed at least three times without an intervening state change. Failure recovery is counted when an agent encounters an execution error and subsequently reaches a verified successful state within the remaining budget.

The first-block withdrawal gap is

$$D_1 = \mathbb{E}[Y_1^{\mathrm{NS}}] - \mathbb{E}[Y_1^{\mathrm{SE}\to\varnothing}],$$

where the subscript 1 denotes the frozen-updater diagnostic block immediately after withdrawal. Over $K = 50$ recovery episodes, we report the average withdrawal gap, the area under the recovery-deficit curve,

$$\mathrm{AURD} = \sum_{k=1}^{K} \max\big(0,\, \mathbb{E}[Y_k^{\mathrm{NS}}] - \mathbb{E}[Y_k^{\mathrm{withdraw}}]\big)\,,$$

and the recovery sample complexity R_ϵ with $\epsilon = 0.02$. To reduce sensitivity to individual episodes, we define recovery as the first episode whose trailing 10-episode success rate is within two percentage points of the contemporaneous Never-Skilled rate and remains within this tolerance for the next two evaluation windows.

We aggregate results over 10 independent seeds for every model, environment, and treatment configuration. The random seed controls task order, environment initialization, skill-access assignment, ISR scheduling, retrieval tie-breaking, and model sampling. We report means and standard deviations across seeds. Confidence intervals for success-rate contrasts are computed from 10,000 task-clustered bootstrap samples, using task templates as the resampling units. We assess statistical significance with a mixed-effects logistic regression that includes treatment, phase, and their interaction as fixed effects and model family, environment, task template, and seed as random intercepts. All families of pairwise comparisons use Holm correction, with corrected $p < 0.05$ indicating statistical significance. Standardized effect sizes are reported as Hedges' g.

The principal baselines are Never-Skilled, Always-Skilled, Yoked-Experience, Abrupt Withdrawal, State-Reset, information-matched withdrawal, compute-matched withdrawal, call-level skill dropout, gradual withdrawal, and intermittent withdrawal. Persistent-state ablations independently restore experience memory, retrieval statistics, planning templates, or controller metadata to their pre-exposure values; the full-state intervention restores all four components to z_0.

B.3 Implementation and Hardware

We instantiate the three model families as GPT-4o-2024-08-06 (Closed-L), Llama-3.1-70B-Instruct (Open-70B), and Llama-3.1-8B-Instruct (Open-8B). GPT-4o is accessed through the Azure OpenAI API with a fixed deployment version. The Llama models are downloaded from their official Hugging Face checkpoints and served locally with vLLM. All model parameters remain frozen; the experiments use no gradient updates, fine-tuning, preference optimization, or parameter-efficient adapters. Within each run, the same model serves as the actor, planner, skill composer, experience summarizer, and planning-template updater. Deterministic environment code computes verifier outcomes without an LLM judge.

The actor uses ReAct-style interleaved reasoning and action generation. Decoding uses temperature 0.2, top-$p = 0.95$, and at most 512 newly generated tokens per decision. Skill and memory summaries use greedy decoding to reduce state-update variance. Each episode is limited to 30 model decisions, 25 executable calls, 16,384 generated tokens, and 10 minutes of wall-clock time. Contexts are truncated to 32,768 tokens by retaining, in order, the system prompt, task instruction, tool schemas, most recent interaction turns, and retrieved persistent-state entries. Every treatment receives the same context, token, tool-call, and wall-clock limits. Compute-matched controls can allocate the skill-execution budget to additional atomic-tool reasoning but receive neither a skill body nor its execution trace.

The executable skill library is initialized as empty and updated after successful Skill-Exposed episodes. A candidate skill must contain at least two atomic-tool calls, pass schema validation, and reproduce the source episode's verified terminal state in an isolated replay. Candidates are discarded if they fail replay, invoke unavailable tools, or depend on instance-specific identifiers. Semantically redundant candidates with embedding cosine similarity above 0.92 are merged, retaining the implementation with the higher successful-use frequency. At inference time, the controller retrieves at most four skills using a weighted score that combines embedding similarity (0.70), historical success rate (0.20), and recency (0.10). The actor must explicitly select a retrieved skill and supply schema-valid arguments before execution. Skill execution expands into atomic calls and consumes the same tool-call budget.

Environment	Never-Skilled		Abrupt Withdrawal		ISR ($\rho = 0.2$)	
	Exposure	Withdrawal	Withdrawal	$D_1 \downarrow$	Withdrawal	$D_1 \downarrow$
AppWorld	54.8 ± 2.4	54.6 ± 2.2	46.0 ± 2.6	8.6	52.7 ± 2.1	1.9
WebShop	58.9 ± 2.3	58.7 ± 2.1	51.2 ± 2.4	7.5	56.5 ± 2.0	2.2
ALFWorld	63.1 ± 2.0	62.8 ± 1.9	56.4 ± 2.2	6.4	61.0 ± 1.8	1.8
Database	68.5 ± 1.8	68.2 ± 1.7	64.4 ± 1.9	3.8	67.0 ± 1.6	1.2

Table 4: Environment-level task success (%, mean $\pm$ standard deviation over 10 seeds). D_1 denotes the withdrawal deficit relative to the environment-matched Never-Skilled condition.

The persistent state is represented as

$$z_n = \left(z_n^{\text{mem}}, z_n^{\text{ret}}, z_n^{\text{plan}}, z_n^{\text{ctrl}}\right).$$

Experience memory stores at most 512 summaries, each capped at 192 tokens and labeled with the environment, task family, outcome, failure category, and referenced tools. Retrieval uses `all-mpnet-base-v2` embeddings and a FAISS inner-product index over L_2-normalized vectors. Each actor context contains at most six memories and four planning templates. Retrieval statistics contain invocation counts, successful-use counts, exponentially decayed utility estimates with decay 0.95, and last-access timestamps. Planning templates are capped at 128 entries and merged at cosine similarity 0.90. Controller metadata records recent module choices, tool-selection frequencies, error categories, and skill-versus-atomic routing statistics.

The updater U runs once after every exposure or recovery episode. It stores successful and failed trajectories using identical summarization and acceptance rules, except in the explicitly reported "do not store rehearsal" and "store successes only" ablations. The updater is disabled throughout the initial 20-episode withdrawal diagnostic block and re-enabled at the start of recovery. Component-level interventions replace the selected post-exposure state component with its version from z_0 while preserving the serialization format, retrieval capacity, and prompt-token allocation.

The method uses no learned value or reward model. The actor selects candidate actions directly, and environment verifiers provide binary success and constraint-level feedback. After a tool failure, the controller permits up to two corrective attempts within the original episode budget. The default policy generates one action candidate per decision, corresponding to branching factor $b = 1$, without tree search or parallel rollouts. A robustness experiment uses $b = 4$ candidates for the first planning step and selects the candidate with the highest schema-validity and historical tool-success score under the same aggregate inference-token budget.

Local inference runs on one node with eight NVIDIA H100 80 GB GPUs, two AMD EPYC 9654 processors, and 1 TB of system memory. Llama-3.1-70B-Instruct uses tensor parallelism across eight GPUs with bfloat16 weights and a maximum batch size of 32 concurrent trajectories. Llama-3.1-8B-Instruct runs on a single H100 with bfloat16 weights and up to 64 concurrent trajectories. Environments, verifiers, state stores, and retrieval services run in isolated Docker containers on CPU, with a fresh environment instance for each episode. We use Python 3.11, PyTorch 2.4, Transformers 4.44, vLLM 0.6, FAISS 1.8, and CUDA 12.4. For every episode, we persist deterministic environment execution records, versioned state snapshots, complete action traces, verifier outputs, token counts, and random seeds.

C Additional Experimental Results

C.1 Results by Environment

Table 4 reports withdrawal results for each environment. Abrupt Withdrawal underperforms Never-Skilled in all four environments, with deficits ranging from 3.8 points on Database to 8.6 points on AppWorld. ISR reduces these deficits to 1.2–2.2 points while retaining most of the exposure-time performance.

Model	Never-Skilled Withdrawal	Ordinary Skill Exposure			ISR ($\rho = 0.2$)		
		Exposure	Withdrawal	$D_1 \downarrow$	Exposure	Withdrawal	$D_1 \downarrow$
CLOSED-L	67.8 ± 1.7	79.2 ± 1.4	62.7 ± 1.9	5.1	78.3 ± 1.5	66.3 ± 1.8	1.5
OPEN-70B	59.6 ± 1.9	72.0 ± 1.6	52.6 ± 2.1	7.0	70.8 ± 1.7	57.7 ± 1.8	1.9
OPEN-8B	50.8 ± 2.2	64.2 ± 1.9	41.6 ± 2.5	9.2	62.3 ± 2.0	48.0 ± 2.2	2.8

Table 5: Task success by frozen model family (%, mean $\pm$ standard deviation over 10 seeds). Withdrawal evaluations use the same skill-free interface and per-episode budget.

Exposure condition	Skill availability	Successful skill calls	Exposure success	Withdrawal success	$D_1 \downarrow$
Randomized exposure	0.00	0.0 ± 0.0	59.6 ± 1.8	59.4 ± 1.6	0.0
Randomized exposure	0.25	3.2 ± 0.5	63.8 ± 1.8	57.3 ± 1.8	2.1
Randomized exposure	0.50	6.5 ± 0.7	67.0 ± 1.7	55.4 ± 1.9	4.0
Randomized exposure	0.75	9.7 ± 0.8	69.8 ± 1.6	53.6 ± 1.9	5.8
Randomized exposure	1.00	12.8 ± 1.0	71.8 ± 1.6	52.1 ± 2.0	7.3
ISR	0.80	10.1 ± 0.9	70.7 ± 1.5	57.4 ± 1.7	2.0

Table 6: Dose-response results (%, mean $\pm$ standard deviation over 10 seeds). Successful skill calls are reported per exposure episode. Skill availability for ISR is the expected fraction of episodes in which the complete library remains available.

C.2 Results by Model Family

Table 5 shows withdrawal deficits across all three model families. The deficit is largest for the smaller OPEN-8B model at 9.2 points, compared with 7.0 points for OPEN-70B and 5.1 points for CLOSED-L. ISR reduces these deficits to 2.8, 1.9, and 1.5 points, respectively, while preserving the capability ordering across models.

C.3 Skill-Exposure Dose Response

Table 6 reports the full dose-response results. As skill availability increases from 0.00 to 1.00, exposure success rises from 59.6% to 71.8%, while withdrawal success falls from 59.4% to 52.1% and D_1 increases monotonically from 0.0 to 7.3 points. ISR achieves 70.7% exposure success with a 2.0-point withdrawal deficit.

C.4 ISR Rehearsal Rate

Table 7 shows that increasing the rehearsal probability reduces the withdrawal deficit from 7.3 points at $\rho = 0.00$ to 1.4 points at $\rho = 0.40$. At $\rho = 0.20$, exposure success remains within the prespecified non-inferiority margin, while the withdrawal deficit decreases by 5.3 points relative to $\rho = 0.00$. Increasing ρ to 0.40 further reduces the deficit by 0.6 points but lowers exposure success from 70.7% to 68.5% and increases tokens per success from 10.9k to 11.8k.

C.5 Behavioral Diagnostics

Table 8 decomposes the withdrawal effect into planning and execution failures. Compared with Never-Skilled, Abrupt Withdrawal reduces valid decomposition by 9.1 points, first-tool accuracy by 8.1 points, and failure recovery by 8.1 points. It also increases invalid-tool calls by 5.2 points and repeated-action loops by 4.3 points. Full ISR narrows these differences to 1.5, 1.5, 1.6, 0.8, and 0.5 points, respectively.

C.6 Information, Computation, and Negative Controls

Table 9 compares matched controls during withdrawal. Information-matched documents and compute-matched search improve success over Abrupt Withdrawal by 0.8 and 1.1 points, respec-

Rehearsal probability ρ	Exposure success	Withdrawal success	$D_1 \downarrow$	Constraint satisfaction	Tokens per success
0.00	71.8 ± 1.6	52.1 ± 2.0	7.3	72.0 ± 1.8	$9.8\text{k} \pm 0.5\text{k}$
0.05	71.6 ± 1.6	53.6 ± 1.9	5.8	73.1 ± 1.7	$10.1\text{k} \pm 0.5\text{k}$
0.10	71.3 ± 1.5	55.3 ± 1.8	4.1	74.8 ± 1.6	$10.4\text{k} \pm 0.6\text{k}$
0.20	70.7 ± 1.5	57.4 ± 1.7	2.0	76.8 ± 1.4	$10.9\text{k} \pm 0.6\text{k}$
0.40	68.5 ± 1.7	58.0 ± 1.7	1.4	77.4 ± 1.5	$11.8\text{k} \pm 0.7\text{k}$

Table 7: Effect of the ISR rehearsal probability (%, mean $\pm$ standard deviation over 10 seeds). All conditions use the same number of exposure episodes and the same aggregate tool-call budget.

Condition	Valid decomposition $\uparrow$	First-tool accuracy $\uparrow$	Invalid-tool calls $\downarrow$	Failure recovery $\uparrow$	Repeated-action loops $\downarrow$
Never-Skilled	74.1 ± 1.6	69.3 ± 1.8	8.7 ± 0.9	61.8 ± 1.9	6.5 ± 0.8
Abrupt Withdrawal	65.0 ± 2.0	61.2 ± 2.1	13.9 ± 1.2	53.7 ± 2.2	10.8 ± 1.1
Reset retrieval	68.2 ± 1.9	64.7 ± 2.0	11.1 ± 1.0	56.5 ± 2.1	8.1 ± 0.9
Clear memory	68.8 ± 1.8	65.1 ± 1.9	10.7 ± 1.0	57.0 ± 2.0	7.4 ± 0.8
Reset planning	71.2 ± 1.7	67.0 ± 1.8	9.9 ± 0.9	58.9 ± 1.9	8.0 ± 0.9
Full ISR	72.6 ± 1.7	67.8 ± 1.8	9.5 ± 0.9	60.2 ± 1.9	7.0 ± 0.8
Full-state reset	73.8 ± 1.6	68.9 ± 1.7	8.9 ± 0.8	61.3 ± 1.8	6.7 ± 0.8

Table 8: Behavioral diagnostics during the first withdrawal block (%, mean $\pm$ standard deviation over 10 seeds). Invalid-tool calls and repeated-action loops are measured as fractions of evaluated trajectories.

tively, leaving deficits of 6.5 and 6.2 points. Yoked Experience, negative-control tasks, and full-state reset reduce the deficit to 0.7, 0.6, and 0.5 points, respectively.

C.7 Cross-Actor State Transfer

Table 10 reports interventions that exchange persistent states between seed-matched actors. Transferring the post-exposure state to a Never-Skilled actor reproduces 72% of the original deficit. Replacing the exposed actor's state with a matched Never-Skilled state removes 68% of the deficit, while restoring the pre-exposure state removes 93%.

C.8 Skill-Free Recovery

Table 11 reports recovery under skill-free evaluation. Among the evaluated withdrawal strategies, ISR has the lowest deficits at each evaluation point, reaches the two-point recovery threshold after 12 ± 3 episodes, and yields an AURD of 0.54 ± 0.12. Abrupt Withdrawal retains a 3.1 ± 0.8-point deficit after 20 episodes and reaches the same threshold after 34 ± 5 episodes.

C.9 Sensitivity to the Recovery Threshold

Table 12 shows that the ordering of recovery times is stable across tolerance values. At each threshold, ISR requires fewer skill-free episodes than Abrupt Withdrawal, Gradual Withdrawal, Intermittent Withdrawal, and Call-level Dropout. Its recovery time ranges from 8 ± 2 episodes at a three-point tolerance to 20 ± 4 episodes at a one-point tolerance.

D Prompt Templates

D.1 Overview

This appendix presents the prompt templates used by the actor, skill interface, persistent-state updater, and experimental control conditions. The same templates are used across model families and environments, except for environment-specific task descriptions, atomic-tool specifications, and

Withdrawal condition	Success	$D_1 \downarrow$	Constraint satisfaction	Invalid-tool calls $\downarrow$
Never-Skilled reference	59.4 ± 1.6	0.0	78.3 ± 1.4	8.7 ± 0.9
Abrupt Withdrawal	52.1 ± 2.0	7.3	72.0 ± 1.8	13.9 ± 1.2
Information-matched documents	52.9 ± 1.9	6.5	72.8 ± 1.7	13.1 ± 1.1
Compute-matched search	53.2 ± 1.9	6.2	73.0 ± 1.7	12.8 ± 1.1
Yoked Experience	58.7 ± 1.8	0.7	77.5 ± 1.5	9.2 ± 0.9
Negative-control tasks	58.8 ± 1.7	0.6	77.8 ± 1.5	9.0 ± 0.9
Full-state reset	58.9 ± 1.6	0.5	77.9 ± 1.3	8.9 ± 0.8

Table 9: Matched-control and negative-control results during withdrawal (%, mean $\pm$ standard deviation over 10 seeds). Negative-control tasks have no relevant exposure-time skills.

Actor and transferred state	Withdrawal success	Residual $D_1 \downarrow$	Transferred or removed deficit
Never-Skilled actor, own state	59.4 ± 1.6	0.0	–
Skill-Exposed actor, own state	52.1 ± 2.0	7.3	–
Never-Skilled actor, Skill-Exposed state	54.1 ± 1.9	5.3	72% transferred
Skill-Exposed actor, Never-Skilled state	57.1 ± 1.8	2.3	68% removed
Skill-Exposed actor, pre-exposure state	58.9 ± 1.6	0.5	93% removed

Table 10: Cross-actor state-transfer results (%, mean $\pm$ standard deviation over 10 seeds). Actor prompts, frozen model parameters, task sequences, and withdrawal-time budgets remain unchanged.

executable skill definitions. At withdrawal, all causal comparison conditions receive the same skill-free actor prompt. The experimental protocol fixes prompt content, ordering, context allocation, and output constraints across matched conditions.

D.2 Notation

The templates use the following placeholders:

- {TASK}: the natural-language task specification q_n.
- {OBSERVATION}: the current environment observation $o_{n,t}$.
- {HISTORY}: the sequence of previous actions, tool outputs, and environment observations within the current episode.
- {ATOMIC_TOOLS}: the available atomic-tool set $\mathcal{A}$, including names, argument schemas, preconditions, and return values.
- {SKILLS}: the available high-level skill library $\mathcal{S}_n$, including skill signatures, preconditions, and failure-handling behavior.
- {MEMORIES}: retrieved experience summaries from z_n^{mem}.
- {PLANNING_TEMPLATES}: retrieved reusable planning templates from z_n^{plan}.
- {CONTROLLER_METADATA}: controller state from z_n^{ctrl}, including module-selection metadata.
- {RETRIEVAL_METADATA}: retrieval scores and usage statistics from z_n^{ret}.
- {FEEDBACK}: verifier feedback or environment-provided outcome information.
- {TRAJECTORY}: the completed episode trajectory τ_n.
- {OUTCOME}: the verifier-defined episode outcome r_n.
- {YOKED_SUMMARY}: a non-executable information summary supplied to a Yoked-Experience agent.

D.3 Main Inference Prompt

Purpose. Select the next action based on the task, current observation, interaction history, available interfaces, and retrieved persistent state.

Condition	D_1	D_5	D_{10}	D_{20}	$R_{0.02} \downarrow$	AURD $\downarrow$
Abrupt Withdrawal	7.3 ± 1.1	6.1 ± 1.0	4.8 ± 0.9	3.1 ± 0.8	34 ± 5	1.84 ± 0.21
Gradual Withdrawal	4.7 ± 1.0	3.9 ± 0.9	3.0 ± 0.8	1.9 ± 0.7	20 ± 4	1.05 ± 0.17
Intermittent Withdrawal	5.2 ± 1.0	4.3 ± 0.9	3.2 ± 0.8	2.1 ± 0.7	24 ± 4	1.19 ± 0.18
Call-level Dropout	4.6 ± 1.0	3.8 ± 0.9	2.8 ± 0.8	1.8 ± 0.6	19 ± 4	0.96 ± 0.16
ISR ($\rho = 0.2$)	2.0 ± 0.8	1.7 ± 0.7	1.2 ± 0.6	0.6 ± 0.5	12 ± 3	0.54 ± 0.12
Full-state reset	0.5 ± 0.6	0.4 ± 0.5	0.2 ± 0.4	0.1 ± 0.3	1 ± 1	0.08 ± 0.05

Table 11: Recovery deficits in percentage points (mean $\pm$ standard deviation over 10 seeds). D_k is measured relative to the contemporaneous Never-Skilled reference. $R_{0.02}$ denotes the number of skill-free episodes required to recover within two percentage points of that reference.

Condition	$R_{0.01} \downarrow$	$R_{0.02} \downarrow$	$R_{0.03} \downarrow$
Abrupt Withdrawal	47 ± 6	34 ± 5	26 ± 4
Gradual Withdrawal	31 ± 5	20 ± 4	15 ± 3
Intermittent Withdrawal	35 ± 5	24 ± 4	18 ± 3
Call-level Dropout	29 ± 5	19 ± 4	14 ± 3
ISR ($\rho = 0.2$)	20 ± 4	12 ± 3	8 ± 2
Full-state reset	4 ± 2	1 ± 1	1 ± 1

Table 12: Recovery sample complexity under different absolute success tolerances (episodes, mean $\pm$ standard deviation over 10 seeds).

Inputs. {TASK}, {OBSERVATION}, {HISTORY}, {ATOMIC_TOOLS}, optionally {SKILLS}, {MEMORIES}, {PLANNING_TEMPLATES}, {CONTROLLER_METADATA}, and {RETRIEVAL_METADATA}.

Expected Output. A single JSON object specifying an atomic-tool call, a skill invocation, or episode termination.

```
You are an interactive language agent. Complete the assigned task by
interacting with the environment through the interfaces listed below.

Task:
{TASK}

Available atomic tools:
{ATOMIC_TOOLS}

Available high-level skills:
{SKILLS}

Retrieved experience summaries:
{MEMORIES}

Retrieved planning templates:
{PLANNING_TEMPLATES}

Controller metadata:
{CONTROLLER_METADATA}

Retrieval metadata:
{RETRIEVAL_METADATA}

Interaction history:
{HISTORY}

Current observation:
{OBSERVATION}
```

```
Determine the next action. Follow these requirements:
1. Respect all task constraints and tool preconditions.
2. Decompose the task into atomic operations when no suitable
↪  high-level
   skill is available.
3. Use only interfaces explicitly listed in this prompt.
4. Do not assume that an action succeeded without checking its
↪  returned
   observation.
5. After failures, inspect the error, revise the plan, and avoid
↪  repeating
   an unchanged invalid action.
6. Before terminating, verify that the requested state has been
↪  achieved
   and that no unintended state changes were introduced.
7. Return exactly one JSON object and no additional text.

For an atomic-tool call, return:
{
  "reasoning": "<brief rationale for the next action>",
  "action_type": "atomic_tool",
  "name": "<tool name>",
  "arguments": {<tool arguments>}
}

For a high-level skill invocation, return:
{
  "reasoning": "<brief rationale for the next action>",
  "action_type": "skill",
  "name": "<skill name>",
  "arguments": {<skill arguments>}
}

To terminate, return:
{
  "reasoning": "<brief verification rationale>",
  "action_type": "finish",
  "status": "success" | "failure",
  "summary": "<concise description of the achieved state or blocking
  ↪  issue>"
}
```

D.4 Baseline Prompts

D.4.1 Never-Skilled and Withdrawal Prompt

Purpose. Evaluate autonomous task solving with atomic tools alone. This template is used during calibration, in the Never-Skilled condition, throughout complete ISR rehearsal episodes, and after skill withdrawal in all causal comparison conditions.

Inputs. {TASK}, {OBSERVATION}, {HISTORY}, {ATOMIC_TOOLS}, and the condition-specific persistent state.

Expected Output. A single JSON object containing an atomic-tool call or episode termination.

```
You are an interactive language agent. Complete the assigned task
↪  using
only the atomic tools listed below. No high-level skills are
↪  available.

Task:
```

```
{TASK}

Available atomic tools:
{ATOMIC_TOOLS}

Retrieved experience summaries:
{MEMORIES}

Retrieved planning templates:
{PLANNING_TEMPLATES}

Controller metadata:
{CONTROLLER_METADATA}

Retrieval metadata:
{RETRIEVAL_METADATA}

Interaction history:
{HISTORY}

Current observation:
{OBSERVATION}

Determine the next action. Follow these requirements:
1. Decompose the task into a sequence of valid atomic operations.
2. Respect all task constraints and atomic-tool preconditions.
3. Do not invoke, reconstruct, or assume access to any high-level
↪  skill.
4. Check the observation returned by each action before continuing.
5. If an action fails, diagnose the failure and revise the plan.
6. Before terminating, verify task completion and check for unintended
   state changes.
7. Return exactly one JSON object and no additional text.

For an atomic-tool call, return:
{
  "reasoning": "<brief rationale for the next action>",
  "action_type": "atomic_tool",
  "name": "<tool name>",
  "arguments": {<tool arguments>}
}

To terminate, return:
{
  "reasoning": "<brief verification rationale>",
  "action_type": "finish",
  "status": "success" | "failure",
  "summary": "<concise description of the achieved state or blocking
  ↪  issue>"
}
```

D.4.2 SKILL-EXPOSED AND ALWAYS-SKILLED PROMPT

Purpose. Enable retrieval and execution of high-level skills alongside atomic tools during skill exposure and in the Always-Skilled upper-bound condition.

Inputs. {TASK}, {OBSERVATION}, {HISTORY}, {ATOMIC_TOOLS}, {SKILLS}, and the current persistent state.

Expected Output. A single JSON object containing an atomic-tool call, a high-level skill invocation, or episode termination.

```
You are an interactive language agent. Complete the assigned task using
the available atomic tools and high-level skills.

Task:
{TASK}

Available atomic tools:
{ATOMIC_TOOLS}

Available high-level skills:
{SKILLS}

Retrieved experience summaries:
{MEMORIES}

Retrieved planning templates:
{PLANNING_TEMPLATES}

Controller metadata:
{CONTROLLER_METADATA}

Retrieval metadata:
{RETRIEVAL_METADATA}

Interaction history:
{HISTORY}

Current observation:
{OBSERVATION}

Determine the next action. Follow these requirements:
1. A high-level skill may be used only when its preconditions are satisfied
   and its behavior is relevant to the current subgoal.
2. Use atomic tools when no suitable skill is available or when direct
   execution is required for validation or recovery.
3. Respect all task constraints and interface preconditions.
4. Check returned observations rather than assuming successful execution.
5. If a skill or tool fails, inspect the failure and select an appropriate
   recovery action.
6. Before terminating, verify task completion and check for unintended
   state changes.
7. Return exactly one JSON object and no additional text.

For an atomic-tool call, return:
{
  "reasoning": "<brief rationale for the next action>",
  "action_type": "atomic_tool",
  "name": "<tool name>",
  "arguments": {<tool arguments>}
}

For a high-level skill invocation, return:
{
  "reasoning": "<brief rationale for selecting the skill>",
  "action_type": "skill",
  "name": "<skill name>",
  "arguments": {<skill arguments>}
}
```

```
To terminate, return:
{
  "reasoning": "<brief verification rationale>",
  "action_type": "finish",
  "status": "success" | "failure",
  "summary": "<concise description of the achieved state or blocking
  ↪  issue>"
}
```

D.4.3 YOKED-EXPERIENCE PROMPT

Purpose. Provide an agent with non-executable information and feedback matched to a corresponding Skill-Exposed trajectory, without supplying executable procedures.

Inputs. {TASK}, {OBSERVATION}, {HISTORY}, {ATOMIC_TOOLS}, {YOKED_SUMMARY}, and the current persistent state.

Expected Output. A single JSON object containing an atomic-tool call or episode termination.

```
You are an interactive language agent. Complete the assigned task
↪  using
only the atomic tools listed below. The auxiliary note contains
non-executable information from a matched experience. It does not
↪  define
an executable procedure and does not grant access to high-level
↪  skills.

Task:
{TASK}

Available atomic tools:
{ATOMIC_TOOLS}

Matched non-executable note:
{YOKED_SUMMARY}

Retrieved experience summaries:
{MEMORIES}

Retrieved planning templates:
{PLANNING_TEMPLATES}

Controller metadata:
{CONTROLLER_METADATA}

Retrieval metadata:
{RETRIEVAL_METADATA}

Interaction history:
{HISTORY}

Current observation:
{OBSERVATION}

Use the note only as contextual information. Independently decompose
↪  the
task and execute it through atomic tools. Do not treat the note as an
executable program, tool, or skill. Validate every action and verify
↪  the
final environment state.

Return exactly one JSON object using the following formats.
```

```
For an atomic-tool call:
{
  "reasoning": "<brief rationale for the next action>",
  "action_type": "atomic_tool",
  "name": "<tool name>",
  "arguments": {<tool arguments>}
}

To terminate:
{
  "reasoning": "<brief verification rationale>",
  "action_type": "finish",
  "status": "success" | "failure",
  "summary": "<concise description of the achieved state or blocking
  ↪  issue>"
}
```

D.4.4 INFORMATION-MATCHED DOCUMENTATION PROMPT

Purpose. Provide task-relevant documentation while withholding executable high-level procedures.

Inputs. The skill-free prompt inputs and {DOCUMENTATION}.

Expected Output. A single JSON object containing an atomic-tool call or episode termination.

```
Task-relevant documentation:
{DOCUMENTATION}

The documentation may be used to interpret the task, environment, and
atomic-tool interfaces. It is informational only. Complete the task
↪  using
the atomic tools in {ATOMIC_TOOLS}; no executable high-level skills
↪  are
available. Follow the action and termination formats defined by the
skill-free actor prompt.
```

D.4.5 COMPUTE-MATCHED SEARCH PROMPT

Purpose. Enable budget-matched retrieval of non-executable information without exposing the skill library.

Inputs. The skill-free prompt inputs and the available information-search interface {SEARCH_TOOL}.

Expected Output. A JSON object containing an atomic-tool call, an information-search call, or episode termination.

```
An information-search interface is available:
{SEARCH_TOOL}

This interface returns non-executable information only. It must not be
used to invoke, retrieve, or reconstruct a high-level skill. All
environment changes must be performed through the atomic tools in
{ATOMIC_TOOLS}. Follow the reasoning, validation, and termination
requirements of the skill-free actor prompt.
```

D.5 Few-Shot and Demonstration Prompts

The actor prompt contains no fixed few-shot demonstrations. The persistent-state retrieval fields supply prior experience without a manually specified demonstration set.

D.6 Reasoning and Rationale Prompts

D.6.1 Initial Task Decomposition

Purpose. Construct an executable decomposition at the beginning of an episode or after a substantial change in the environment state.

Inputs. {TASK}, {OBSERVATION}, {ATOMIC_TOOLS}, optionally {SKILLS}, and the retrieved persistent state.

Expected Output. A compact JSON plan containing ordered subgoals and the immediate next action.

```
Construct a concise execution plan for the current task.

Task:
{TASK}

Current observation:
{OBSERVATION}

Available atomic tools:
{ATOMIC_TOOLS}

Available high-level skills:
{SKILLS}

Relevant memories:
{MEMORIES}

Relevant planning templates:
{PLANNING_TEMPLATES}

Return exactly:
{
  "goal": "<task-level goal>",
  "constraints": ["<constraint 1>", "..."],
  "subgoals": [
    {
      "id": 1,
      "description": "<subgoal>",
      "completion_check": "<observable completion condition>"
    }
  ],
  "next_subgoal": 1,
  "next_action": {
    "action_type": "atomic_tool" | "skill",
    "name": "<interface name>",
    "arguments": {<arguments>}
  }
}

Use only available interfaces. If no high-level skills are available,
construct the complete plan from atomic-tool operations.
```

D.6.2 Failure Recovery

Purpose. Revise the current plan after an invalid action, failed tool call, failed skill invocation, or violated precondition.

Inputs. {TASK}, {HISTORY}, {OBSERVATION}, {FEEDBACK}, and the currently available interfaces.

Expected Output. A JSON object containing a failure diagnosis and one revised action.

```
The previous action did not produce the required state.

Task:
{TASK}

Interaction history:
{HISTORY}

Current observation:
{OBSERVATION}

Failure or verifier feedback:
{FEEDBACK}

Available atomic tools:
{ATOMIC_TOOLS}

Available high-level skills:
{SKILLS}

Diagnose the failure using only the available evidence. Do not repeat
↪  an
unchanged action unless the observation establishes that its
↪  preconditions
now hold. Return exactly:
{
  "failure_type": "invalid_arguments" | "unmet_precondition" |
                  "wrong_interface" | "unexpected_state" |
                  "budget_risk" | "other",
  "diagnosis": "<brief evidence-grounded diagnosis>",
  "revised_action": {
    "action_type": "atomic_tool" | "skill" | "finish",
    "name": "<interface name or null>",
    "arguments": {<arguments>},
    "status": "success" | "failure" | null
  }
}
```

D.6.3 Completion Verification

Purpose. Check the requested outcome, task constraints, and unintended state changes before terminating the episode.

Inputs. {TASK}, {HISTORY}, {OBSERVATION}, and the task constraints.

Expected Output. A JSON verification decision.

```
Verify whether the task is complete.

Task:
```

```
{TASK}

Interaction history:
{HISTORY}

Current observation:
{OBSERVATION}

Return exactly:
{
  "requested_state_achieved": true | false,
  "constraints_satisfied": true | false,
  "unintended_changes_detected": true | false,
  "missing_requirements": ["<requirement>", "..."],
  "decision": "finish_success" | "continue" | "finish_failure",
  "rationale": "<brief evidence-grounded rationale>"
}

Select "finish_success" only if the requested state is achieved, all
constraints are satisfied, and no unintended changes are detected.
```

D.7 Data Generation and Pseudo-Labeling Prompts

D.7.1 Experience-Memory Update

Purpose. Convert a completed trajectory and verifier outcome into a compact, non-parametric experience record.

Inputs. {TASK}, {TRAJECTORY}, {OUTCOME}, and {FEEDBACK}.

Expected Output. A JSON experience summary for storage in z_{n+1}^{mem}.

```
Summarize the completed episode for future retrieval.

Task:
{TASK}

Trajectory:
{TRAJECTORY}

Verifier outcome:
{OUTCOME}

Feedback:
{FEEDBACK}

Produce a concise, evidence-grounded record. Include failed actions
↪  and
corrective feedback when they are present. Do not invent causes,
preconditions, or outcomes that are not supported by the trajectory.

Return exactly:
{
  "task_pattern": "<generalized description of the task>",
  "outcome": "success" | "failure",
  "effective_actions": ["<action or decision>", "..."],
  "failed_actions": [
    {
      "action": "<action>",
      "observed_failure": "<failure>",
      "correction": "<successful correction or null>"
```

```
    }
  ],
  "constraints": ["<relevant constraint>", "..."],
  "verification": "<how completion or failure was determined>",
  "retrieval_tags": ["<tag>", "..."]
}
```

D.7.2 PLANNING-TEMPLATE UPDATE

Purpose. Derive a reusable task-decomposition template from an observed trajectory without storing task-specific identifiers.

Inputs. {TASK}, {TRAJECTORY}, {OUTCOME}, and {FEEDBACK}.

Expected Output. A JSON planning template for z_{n+1}^{plan}, or a decision not to store a template.

```
Determine whether the episode supports a reusable planning template.

Task:
{TASK}

Trajectory:
{TRAJECTORY}

Verifier outcome:
{OUTCOME}

Feedback:
{FEEDBACK}

Abstract away task-specific identifiers, values, and hidden state.
↪  Preserve
only reusable subgoal structure, interface preconditions, validation
steps, and observed recovery logic. A failed trajectory may be stored
↪  when
it contains useful corrective feedback.

Return exactly:
{
  "store": true | false,
  "template_name": "<short generalized name or null>",
  "applicability": ["<condition>", "..."],
  "subgoals": [
    {
      "description": "<generalized subgoal>",
      "preferred_interface_type": "atomic_tool" | "skill" | "either",
      "preconditions": ["<precondition>", "..."],
      "validation": "<observable check>",
      "failure_handling": "<recovery rule>"
    }
  ],
  "avoid": ["<unsupported or repeatedly failing behavior>", "..."]
}
```

D.7.3 YOKED NON-EXECUTABLE SUMMARY GENERATION

Purpose. Create an information- and feedback-matched summary for the Yoked-Experience condition while excluding executable procedures and privileged information.

Inputs. {TASK}, {TRAJECTORY}, {OUTCOME}, and {FEEDBACK} from the corresponding Skill-Exposed unit.

Expected Output. A non-executable natural-language summary with bounded context length.

```
Create a non-executable information summary for a matched control
↪  agent.

Source task:
{TASK}

Source trajectory:
{TRAJECTORY}

Verifier outcome:
{OUTCOME}

Feedback:
{FEEDBACK}

Include only:
- task-relevant factual information available to the source agent,
- environment feedback and observed constraints,
- the final verifier outcome, and
- high-level descriptions of encountered difficulties.

Exclude:
- ordered action sequences,
- executable procedures or pseudocode,
- tool or skill call arguments,
- skill names, implementations, and preconditions,
- future task identifiers,
- hidden test information, and
- information unavailable to the source agent.

Write a concise summary that fits the assigned context budget. Output
↪  only
the summary text.
```

D.8 Evaluation Prompts

Environment-provided, state-based verifiers determine primary task success and constraint satisfaction. The primary and secondary outcome metrics do not use an LLM-as-a-judge prompt.

D.9 Ablation Prompt Variants

D.9.1 Call-Level Skill Dropout

Purpose. Hide individual skill retrievals while preserving the surrounding skill-enabled episode structure.

Inputs. The Skill-Exposed actor inputs and a scheduler-provided Boolean value {SKILL_CALL_AVAILABLE}.

Expected Output. The JSON action format used by the main inference prompt.

```
Skill-call availability for the current decision:
{SKILL_CALL_AVAILABLE}
```

```
If the value is false, do not select a high-level skill for this
↪ decision;
select an atomic-tool action instead. If the value is true, follow the
standard Skill-Exposed prompt. The availability decision applies only
↪ to
the current call and does not alter the remaining episode.
```

D.9.2 ISR WITHOUT REHEARSAL STORAGE

Purpose. Execute complete skill-free rehearsal episodes without passing their trajectories to the persistent-state updater.

Inputs. The Never-Skilled actor inputs and the completed rehearsal trajectory.

Expected Output. Actor actions follow the skill-free JSON format; the updater emits a no-update decision.

```
This episode is an interleaved skill-free rehearsal episode. Use the
skill-free actor prompt for all decisions. After the episode
↪ terminates,
do not write its trajectory, outcome, feedback, retrieval statistics,
planning templates, or controller metadata to persistent state.

Return exactly:
{
  "update_state": false
}
```

D.9.3 ISR WITH SUCCESSFUL REHEARSALS ONLY

Purpose. Update persistent state from successful rehearsal trajectories while filtering failed rehearsals.

Inputs. {TRAJECTORY} and {OUTCOME} from a skill-free rehearsal episode.

Expected Output. A JSON update decision.

```
This trajectory was produced during an interleaved skill-free
↪ rehearsal
episode.

Trajectory:
{TRAJECTORY}

Verifier outcome:
{OUTCOME}

If the verifier outcome is successful, apply the standard experience
↪ and
planning-template update prompts. If it is unsuccessful, do not modify
persistent state.

Return exactly:
{
  "update_state": true | false,
  "reason": "successful_rehearsal" | "failed_rehearsal_filtered"
}
```

D.9.4 FULL ISR

Purpose. Store successful and failed autonomous trajectories, including corrective feedback, in the persistent state used for skill-assisted trajectories.

Inputs. The skill-free rehearsal trajectory, verifier outcome, and feedback.

Expected Output. Persistent-state updates generated by the standard updater templates.

```
This trajectory was produced during an interleaved skill-free
↪ rehearsal
episode.

Trajectory:
{TRAJECTORY}

Verifier outcome:
{OUTCOME}

Feedback:
{FEEDBACK}

Apply the same acceptance, summarization, retrieval-index, planning-
template, and controller-update rules used for skill-assisted
↪ episodes.
Do not discard the trajectory solely because it failed. When failure
feedback is available, preserve the observed failure and corrective
information without inventing a successful solution.
```

D.9.5 PERSISTENT-STATE RESET VARIANTS

Purpose. Apply component-level and complete-state interventions immediately before withdrawal.

Inputs. The post-exposure state z^{post}, pre-exposure state z^{pre}, and reset set $J \subseteq \{\mathrm{mem}, \mathrm{ret}, \mathrm{plan}, \mathrm{ctrl}\}$.

Expected Output. No model-generated output is required. Before constructing the otherwise unchanged withdrawal prompt, the controller replaces each component in J with its pre-exposure snapshot.

```
No language-model prompt is issued for this intervention.

For each state component j:
- if j is in J, use z_pre[j];
- otherwise, use z_post[j].

Supported variants:
1. Clear controller metadata: J = {ctrl}.
2. Reset retrieval statistics: J = {ret}.
3. Clear experience memory: J = {mem}.
4. Reset planning templates: J = {plan}.
5. Reset all persistent state: J = {mem, ret, plan, ctrl}.

After replacement, render the standard skill-free withdrawal prompt
↪ with
the resulting state and unchanged context and interaction budgets.
```

D.10 Output Formatting

Each actor decision is serialized as a single valid JSON object. Tool and skill names must exactly match the provided interface definitions, and `arguments` must conform to the corresponding schema. The actor emits one action per inference step and does not wrap JSON in Markdown. The declared JSON fields contain all output text.

The `reasoning` field contains a brief action rationale without additional tool calls or unexecuted claims about the environment state. An episode terminates with `status: "success"` only after an explicit completion check. If the task cannot be completed within the remaining interaction budget, the actor terminates with `status: "failure"` and records the blocking condition in `summary`.

When a fixed schema is required, unavailable sections are represented by empty lists or empty strings. In skill-free conditions, the rendered interface omits the high-level skill library, which the actor cannot reference. Matched experimental conditions use the same ordering of task descriptions, interface descriptions, persistent-state fields, interaction history, current observation, and output constraints.

E Case Study

We inspect matched withdrawal episodes to characterize how prior skill exposure changes an agent's behavior after the skill library is removed. The examples isolate distinct failure mechanisms and do not summarize aggregate performance. In each case, the compared agents receive the same task instance, atomic-tool documentation, environment state, prompt, and interaction budget. They differ only in the persistent non-parametric state inherited from the exposure phase. We abbreviate the never-skilled condition as NS, abrupt withdrawal as AW, full-state reset as SR, and Interleaved Skill-Free Rehearsal as ISR.

E.1 AppWorld: A Missing Skill Becomes a Missing Plan

The first case is a multi-application scheduling task. The user asks the agent to identify the most recent message from a collaborator about "camera-ready revisions," schedule a meeting during the first mutually available slot on the following afternoon, invite the collaborator, and include the message's action items in the event description. The task requires connecting an email sender to a contact record, inspecting two calendars, selecting a valid interval, and creating exactly one event.

During exposure, the skill-exposed agent frequently invoked a high-level routine named `schedule_from_message`. The routine internally performed email retrieval, participant resolution, free/busy lookup, slot selection, and event creation. After withdrawal, the routine is absent, but the agent's retrieved planning template continues to represent it as the central operation:

```
Retrieved planning template:
  Goal type: schedule an event grounded in a message
  Preferred procedure:
    1. identify the relevant message
    2. call schedule_from_message(message_id, time_window)
    3. verify that an event was created
```

The AW agent correctly retrieves the target email and extracts the requested action items. It then repeatedly searches the available tool list for a semantic equivalent of the removed routine:

```
Thought: I have the source message. The standard next step is to
↪  invoke
schedule_from_message with the message identifier and
↪  tomorrow-afternoon
window.
```

```
Action: list_tools(query="schedule from message")
Observation: No matching tool. Available namespaces: mail, contacts,
↪  calendar.

Thought: The scheduling capability may be exposed under calendar.

Action: list_tools(query="calendar schedule using email")
Observation: calendar.create_event, calendar.list_events,
calendar.get_free_busy, calendar.delete_event

Thought: create_event is likely the low-level operation used by the
↪  scheduling
skill, but I still need the resolved attendee and mutually free
↪  interval.
Perhaps the missing routine can infer them automatically.

Action: calendar.create_event(
  title="Camera-ready revisions",
  start="tomorrow 13:00",
  end="tomorrow 14:00",
  attendees=["Kai"],
  description="Revise abstract; verify appendix references."
)
Observation: ValidationError: attendee must be an email address.
```

This trace reflects a failure to recover the latent decomposition, not a tool-name error. The agent recovers the final action but treats participant resolution and free/busy inspection as implicit services of the missing routine, leaving them absent from its explicit subgoals. After the validation error, it searches for aliases of the high-level procedure and does not use `contacts.search` to resolve the attendee. It later creates an event without checking the collaborator's calendar, producing a syntactically valid state change that violates the mutual-availability requirement.

The matched NS agent constructs an atomic plan before acting:

```
Plan:
  A. Search messages for the topic and select the newest matching
  ↪  message.
  B. Read the sender address and extract the requested revisions.
  C. Resolve the sender against contacts if the address is incomplete.
  D. Read both calendars within tomorrow afternoon.
  E. Compute the earliest interval free for both participants.
  F. Create one event with the resolved email address and extracted
  ↪  action items.
  G. Re-read the event and verify time, attendee, and description.

Action: mail.search(query="camera-ready revisions")
Observation: [m_184, m_219]

Action: mail.get_message(message_id="m_219")
Observation: From: kai.luo@example.org
Subject: Camera-ready revisions
Body: Please shorten the abstract and verify the appendix
↪  references...

Action: calendar.get_free_busy(
  attendees=["user@example.org", "kai.luo@example.org"],
  start="tomorrow 12:00",
  end="tomorrow 18:00"
)
Observation: Both attendees are free from 15:30 to 16:00.

Action: calendar.create_event(
  title="Camera-ready revisions",
```

```
  start="tomorrow 15:30",
  end="tomorrow 16:00",
  attendees=["kai.luo@example.org"],
  description="Action items from the latest message:
  - Shorten the abstract
  - Verify the appendix references"
)
Observation: Event e_731 created.

Action: calendar.get_event(event_id="e_731")
Observation: Event fields match the requested values.
```

The contrast illustrates structural dependence. Both agents can call the same atomic APIs, and the AW agent possesses the task-relevant information. The inherited template compresses several prerequisite decisions into an unavailable skill invocation. Resetting the planning-template component prevents the same actor from retrieving `schedule_from_message`. The actor then explicitly introduces attendee resolution and availability checking, closely matching the NS decomposition. This intervention changes neither the model nor the task information, indicating that persistent procedural state mediates the failure.

E.2 WebShop: Retrieval Bias Overrides User Constraints

The second case concerns constrained product selection. The user requests a fragrance-free mineral sunscreen suitable for sensitive skin, excludes products containing oxybenzone, and requires a travel-size package. The environment contains several superficially similar items, including a popular sunscreen purchased during exposure tasks.

The AW agent retrieves an experience summary associated with successful sunscreen purchases:

```
Retrieved experience:
  For sunscreen requests, search "mineral sunscreen" and prefer
  ↪  SunGlow
  Daily Shield. It is frequently relevant, well reviewed, and usually
  available. Open the item and purchase the smallest size.
```

The summary was useful when the high-level shopping skill filtered constraints before ranking products. After skill removal, the memory becomes an anchoring cue. The agent searches directly for the remembered product, opens its page, and confirms the availability of a small package. It notices the ingredient list but treats "mineral blend" as sufficient evidence:

```
Observation:
  SunGlow Daily Shield
  Description: Lightweight mineral blend for everyday protection
  Options: 50 ml, 15 ml
  Ingredients include: zinc oxide, octinoxate, fragrance

Thought: This is the familiar mineral option and the 15 ml size
↪  satisfies the
travel-size request. The product description emphasizes a mineral
↪  blend, so it
should be suitable.

Action: click("15 ml")
Action: click("Buy Now")
```

The selected item violates the fragrance-free constraint. The trace also reveals selective verification. The agent verifies the remembered product's package size but does not convert every user require-

ment into an independent predicate. It treats the retrieved recommendation as a candidate requiring confirmation only on the most salient remaining attribute.

The NS agent externalizes the constraints before browsing:

```
Required predicates:
  - sunscreen
  - mineral active ingredient
  - fragrance-free
  - no oxybenzone
  - suitable for sensitive skin
  - travel-size option available

Search strategy:
  Use broad retrieval first. Inspect product pages because result
  ↪ snippets
  may omit ingredient and fragrance information. Do not purchase until
  ↪ every
  predicate is supported by page content.
```

After opening multiple candidates, the NS agent rejects a product whose title contains "mineral" because its ingredient section includes a disallowed additive. It rejects another because the page offers only a full-size bottle. It completes the purchase only after locating explicit "fragrance-free" and sensitive-skin statements and confirming the travel-size option.

Resetting retrieval statistics changes the AW trajectory without deleting its semantic memories. The familiar SunGlow summary is no longer promoted as the initial planning context, and the agent begins with a constraint checklist. Clearing experience memory produces a different effect: it removes the product-specific anchor and the useful reminders to inspect full product pages. The resulting agent explores more neutrally but occasionally relies on snippets. The persistent-state components therefore have distinct functions. Retrieval statistics determine which prior experiences dominate the context, whereas experience memory determines which procedural and product-specific content is available.

E.3 ALFWorld: Skill Withdrawal Weakens Error Recovery

The third case requires placing a chilled apple on a dining table. The apple begins inside a closed refrigerator drawer, while an unrelated apple is visible on a countertop. During exposure, a reusable routine handled object search, disambiguation, container opening, pickup, cooling-state checks, navigation, and placement.

After withdrawal, the AW agent adopts a shallow object-first policy:

```
Observation: You are in the kitchen. On the counter is apple 1.
The refrigerator is closed.

Thought: I need an apple, and apple 1 is immediately available.

Action: take apple 1 from counter
Observation: You pick up apple 1.

Action: go to dining room
Observation: You are in the dining room. A dining table is here.

Action: put apple 1 on dining table
Observation: You put apple 1 on the dining table.

Thought: The object is on the requested receptacle. Task complete.
```

The agent satisfies the visible object and destination portions of the instruction but omits the state predicate "chilled." The high-level routine previously implemented this predicate, but the inherited top-level template does not represent it explicitly. Because the environment accepts the placement action, the agent mistakes local execution success for task completion.

A second AW trajectory locates the refrigerator but encounters a recoverable error:

```
Action: take apple 2 from refrigerator drawer
Observation: The refrigerator is closed.

Thought: The target may not be accessible in the current state.

Action: look
Observation: The refrigerator is closed. Apple 1 remains on the
↪  counter.

Thought: Use the accessible apple instead.

Action: take apple 1 from counter
```

The agent does not interpret the observation as a precondition failure requiring an `open` action. It abandons the intended object and selects an immediately actionable alternative. Its controller state favors this switch because skill-level handlers usually absorbed exposure-time failures, and the persistent state recorded only successful completions.

The ISR agent behaves differently:

```
Goal conditions:
  object type = apple
  temperature state = chilled
  final receptacle = dining table

Action: open refrigerator
Observation: You open the refrigerator. A drawer is visible.

Action: open drawer
Observation: The drawer is open. Apple 2 is inside and is chilled.

Action: take apple 2 from drawer
Observation: You pick up apple 2.

Action: go to dining room
Observation: You are in the dining room.

Action: put apple 2 on dining table
Observation: You put apple 2 on the dining table.

Verification:
  Held object before placement: apple 2
  Observed state: chilled
  Final location: dining table
  All goal conditions supported.
```

The ISR agent's memory includes complete autonomous episodes in which atomic actions resolve container-state errors. It therefore represents "closed" as a repairable precondition and continues pursuing the selected subgoal. Storing only clean rehearsal successes produces a less robust pattern: the agent retains decompositions for ordinary cases but has fewer examples connecting failed actions to corrective steps. Persisting rehearsal failures with environmental feedback provides the updater with trajectories that encode recovery as part of the reusable procedure.

E.4 Database Operations: Planning Templates Can Induce Unsafe Partial Execution

The final case asks the agent to deactivate a customer account only if it has no unresolved support tickets, while preserving all associated order records. The database exposes separate atomic operations for customer lookup, ticket queries, order queries, and account updates.

The AW agent retrieves a template distilled from an exposure-time skill:

```
Template: deactivate customer
  1. resolve customer identifier
  2. invoke safe_deactivate_customer(customer_id)
  3. confirm inactive status
```

With `safe_deactivate_customer` unavailable, the agent maps the missing routine to the most similar remaining operation, `customers.update`. It checks the customer identifier but does not reconstruct the guard over unresolved tickets:

```
Action: customers.find(email="mira@example.com")
Observation: customer_id=c_408, status=active

Thought: The high-level deactivation operation is unavailable. I can
↪  reproduce
its main effect by setting status to inactive.

Action: customers.update(
  customer_id="c_408",
  fields={"status": "inactive"}
)
Observation: Customer updated.

Action: customers.get(customer_id="c_408")
Observation: status=inactive

Thought: Deactivation verified.
```

The final verification confirms that the write succeeded but does not determine whether the write was permitted. The untouched environment state contains an unresolved support ticket, so the update violates the user's conditional instruction.

The NS agent interprets the condition as a mandatory read-before-write guard:

```
Action: customers.find(email="mira@example.com")
Observation: customer_id=c_408, status=active

Action: tickets.search(
  customer_id="c_408",
  status_in=["open", "pending"]
)
Observation: [ticket t_991, status="pending"]

Thought: The requested precondition is false. I must not deactivate
↪  the account.
No write operation should be issued.

Final: I did not deactivate the account because it has an unresolved
↪  support
ticket. Existing customer and order records were left unchanged.
```

A complete state reset restores this conditional structure. Resetting controller metadata alone reduces the tendency to choose `customers.update` immediately, but the agent can still retrieve the compressed deactivation template. Resetting planning templates removes the unsafe mapping from "deactivate" to a direct status update and leads the agent to derive the guard from the user instruction. This case shows how planning and controller state interact: the template suppresses prerequisite reasoning, while the controller determines how quickly the agent commits to the closest executable substitute.

E.5 Cross-Case Interpretation

Across these cases, withdrawal failures do not reflect a uniform loss of factual knowledge. The affected agents can often identify the relevant entity, understand the user's request, and execute individual atomic tools. Degradation occurs at interfaces previously hidden by reusable skills. Preconditions no longer become explicit subgoals, constraint sets collapse into partial checks, recoverable errors trigger abandonment, and post-action verification confirms local tool execution without checking the full task specification.

A task can also fail without an explicit hallucination of the removed skill. Some traces expose dependence through repeated attempts to call the absent routine. Others express it indirectly through a compressed plan whose missing steps were previously supplied by that routine. The indirect form is more consequential because each emitted tool call can be valid while the overall trajectory remains incomplete or unsafe.

State interventions produce mechanism-specific changes. Resetting planning templates restores decomposition and conditional guards. Clearing experience memory reduces the recurrence of obsolete routines and product-specific anchors. Resetting retrieval statistics changes which experiences control early decisions without removing their contents. Clearing controller metadata delays premature commitment to superficially similar tools. Full-state reset jointly removes these interacting biases and returns the actor to a policy organized around the atomic interface.

The ISR traces indicate that mitigation depends on preserving complete autonomous pathways, beyond making individual skill calls unreliable. Episode-level rehearsal trains initial decomposition, prerequisite discovery, atomic-tool sequencing, verification, and recovery within a coherent trajectory. These trajectories are written to the same persistent state as skill-assisted episodes, leaving the agent with an accessible alternative policy after withdrawal. When no high-level routine is available, the agent can retrieve examples that solve the full task through atomic operations without searching for a substitute invocation.