EXECUTE WHAT REMAINS: STATE-GROUNDED SKILL EXECUTION FOR LANGUAGE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Language agents can store reusable procedures yet fail to execute them effectively on new tasks. An admissible source action need not advance the current task, and skipping actions with satisfied effects does not determine which stage should proceed next. We introduce State-Grounded Skill Execution (SG), which separates persistent skill knowledge from episode-local submitted progress. SG reconstructs candidates for the remaining acquisition, transformation, and delivery stages, prioritizes direct stage actions, and preserves progress across bounded yields to an incumbent policy. In a prospective comparison on 120 ALFWorld train tasks with no exact-task overlap with prior experiments, all executors use the same skill bank, receiver, and initial budgets. SG improves full-task success over an effect-aware source-order executor (EA) from 30.00% to 46.67%, with 20 SG-only successes and no EA-only success. Mean priced runtime and wall-clock time decrease by 23.16% and 23.18%, respectively, alongside fewer model calls, environment actions, and tokens. On the complete official `valid_unseen` split, the fixed SG portfolio improves success over the incumbent from 20.90% to 30.60%. The largest gains occur on cleaning tasks that require an intermediate transformation. These results show that controlling a stored procedure's remaining execution can improve both task success and execution efficiency.

1 INTRODUCTION

Language agents increasingly turn successful interactions into reusable trajectories, workflows, and procedural skills (Wang et al., 2024; 2025; Zheng et al., 2025; Mi et al., 2026; Zhang et al., 2026). Retrieval is only the beginning of reuse: after selecting a skill, the agent must decide which source operations remain useful in the current task. We study this post-retrieval control problem.

The difficulty is a mismatch between the state in which a procedure was recorded and the state in which it is reused. An action inherited from the source skill may remain executable even when it no longer advances the current task. Conversely, observing that a target effect is already satisfied does not reveal whether the agent should acquire an object, perform a transformation, or deliver it. Effective reuse must therefore distinguish three signals that source-order execution tends to conflate:

$$\text{action admissibility} \neq \text{effect completion} \neq \text{required execution stage}. \tag{1}$$

This distinction is especially important for multi-stage procedures. A stateful executor can remember its source position, and an effect-aware variant can skip a transformation whose effect is already visible. Both still advance through a fixed source order rather than reconstructing what remains necessary.

We introduce *State-Grounded Skill Execution* (SG), a bounded controller over an incumbent policy (Figure 1). A persistent skill ω stores source-supported operations, roles, dependencies, and order, while transient state z_t records current bindings and submitted nodes. SG combines these objects with public observations and admissible commands to infer the residual acquisition, transformation, or delivery stage and schedule an unsubmitted source operation within it. When no skill action is available, the controller preserves z_t and yields temporarily to the incumbent. The procedure remains fixed while its execution follows what remains to be done.

We evaluate execution while holding the receiver, skill bank, public-state interface, action permissions, and budgets fixed. In the 120-task prospective comparison, SG solves 56 tasks versus 36 for EA; all

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.

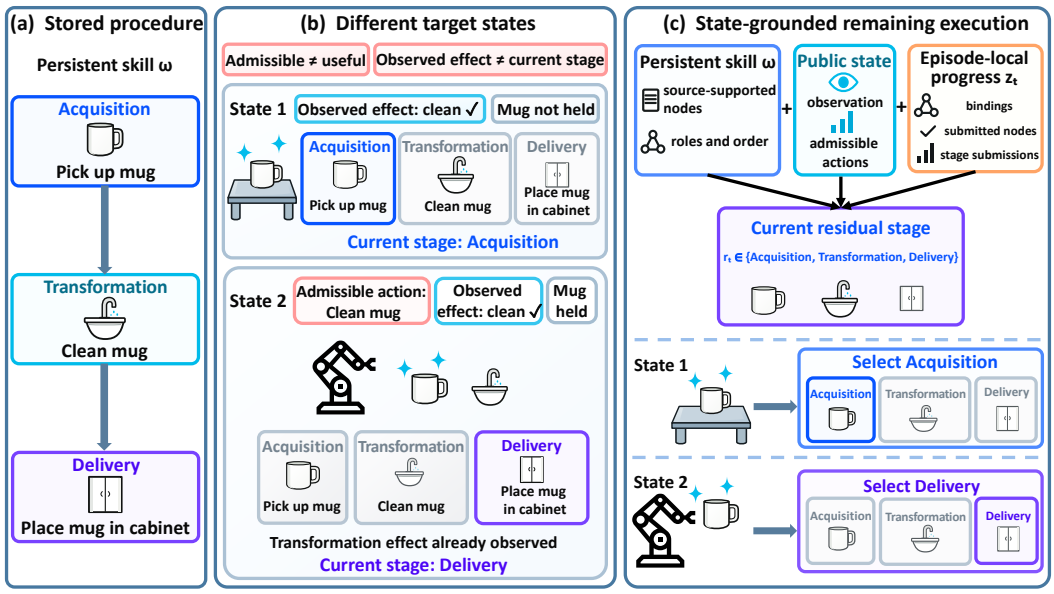


Figure 1: Possessing a procedure does not specify its remaining execution. Action admissibility, publicly observed effect completion, and the stage currently required by the task are distinct control signals. Colored outlines indicate the selected current stage; gray stages are not currently selected.

20 discordant pairs favor SG. Mean priced runtime decreases by 23.16%. On the complete official `valid_unseen` split, SG solves 41/134 tasks versus 28/134 for the incumbent. The detailed task and resource results appear in Section 6.

Our contributions are threefold:

- We formulate procedural-skill reuse as control of the execution that remains under the current public state and invocation-local progress.
- We introduce SG, which reconstructs stage-specific candidates from unsubmitted source nodes, prioritizes direct actions, and supports bounded yield and resume.
- Matched-skill and complete-split evaluations show higher full-task success with lower execution-resource use, with the largest gains on the evaluated cleaning tasks.

2 Related Work

Agent memory and procedural skills. Language-agent memory has evolved from raw trajectories to reusable workflows and executable skills. Voyager grows verified programs from embodied interaction (Wang et al., 2024); AWM induces workflows from demonstrations or online experience and identifies missing intermediate observations as a source of execution inflexibility (Wang et al., 2025); SkillWeaver discovers API-like skills for web agents (Zheng et al., 2025); and Skill-Pro conditions action generation on the current state and active skill (Mi et al., 2026). We fix the procedural asset and isolate its execution control.

Memory selection and skill composition. MemRL learns item-level utility for retrieved episodic memories (Zhang et al., 2026), while Compositional Skill Routing decomposes requests and composes multiple skills through dependency-aware plans (Gao, 2026). These methods decide which memories to use or how to compose them. SG instead controls the remaining operations after a finite skill has been selected.

Temporal abstraction and reactive control. Options model temporally extended behavior with initiation and termination conditions (Sutton et al., 1999; Bacon et al., 2017); behavior trees provide modular reactive control, and memoryful sequences retain their execution position (Colledanchise

et al., 2019; BehaviorTree.CPP, 2026). SG applies these established primitives to a source-derived procedure through submitted-node memory, residual-stage candidates, direct-first scheduling, and bounded interleaving with an incumbent. The contribution is this residual-execution representation and its controlled full-task evaluation under a fixed skill bank.

Test-time control. Value-of-computation methods trade decision improvement against computation cost (Hay et al., 2012), whereas test-time training adapts model parameters at inference time (Sun et al., 2020). SG keeps parameters fixed and adapts only the execution of an existing skill through transient controller state.

3 Problem Formulation

We study a language agent that already owns a finite library of procedural skills and must reuse them on new tasks. Skill construction and any parameter training precede evaluation. The skill bank and selection rules remain fixed during evaluation, while task-conditioned lookup and incumbent retrieval follow those rules. The online problem is to decide when an existing skill should intervene, what it should execute next, and when control should return to the incumbent policy.

Public task state. At step t, the environment has latent state s_t, public observation o_t, goal g, and admissible actions $\mathcal{A}_t$. The agent cannot access s_t. A frozen interpreter instead constructs

$$\chi_t = (g, o_t, \mathcal{A}_t), \qquad \hat{s}_t = \phi(\chi_t), \tag{2}$$

where $\hat{s}_t$ contains only facts recoverable from these public inputs. All executors use the same public interface and initial budgets, but maintain their own interaction histories and remaining resources. Invoking a skill resets neither the environment nor that policy's history or consumed budget.

Persistent skill knowledge. Let π_0 be the incumbent policy and $\mathcal{K}$ a skill bank frozen before evaluation. A skill

$$\omega = (\mathcal{O}_\omega, \sigma_\omega, \mathcal{E}_\omega, \mathcal{B}_\omega, \iota_\omega, \beta_\omega) \tag{3}$$

stores source-supported operations, their source order and public effects, role bindings, and activation and return conditions. It contains reusable procedural knowledge but no execution position for the current episode. We hold π_0, $\mathcal{K}$, the receiver, accessible information, permissions, and initial budgets fixed across executors.

Episode-local progress. Each skill invocation creates transient state z_t: current bindings, submitted source nodes, option visits, acquisition/transformation/delivery submissions, repair waits, and completion or failure flags. Public facts remain in $\hat{s}_t$; records in z_t are scheduling memory about what the controller has issued, not assertions about latent environment truth. The state is reset at the end of the episode.

The executor chooses among continuing the incumbent, invoking a skill, submitting an admissible skill action, temporarily yielding to the incumbent while preserving z_t, permanently returning control, or terminating. Delivery is a task stage and is distinct from returning control. Full-task success $Y(\tau) \in \{0, 1\}$ is determined only by the environment verifier on complete trajectory τ; internal controller completion cannot replace it.

We measure calls, environment actions, input/output tokens, GPU and CPU time, and wall time. The preregistered aggregate resource unit is

$$C_{\text{run}}(\tau) = 0.01 T_{\text{GPU}} + 0.001 T_{\text{CPU}}, \tag{4}$$

reported together with every component. The fixed-bank objective is paired: improve verifier-defined success over a comparator on the same tasks while reducing, or at least not increasing, complete-trajectory execution resources. Acquisition costs are analyzed separately from serving an already-owned skill bank.

4 State-Grounded Skill Execution

State-Grounded Skill Execution converts a stored source procedure into the bounded fragment that remains useful in the current task. Given skill ω, public state $\hat{s}_t$, admissible actions $\mathcal{A}_t$, and

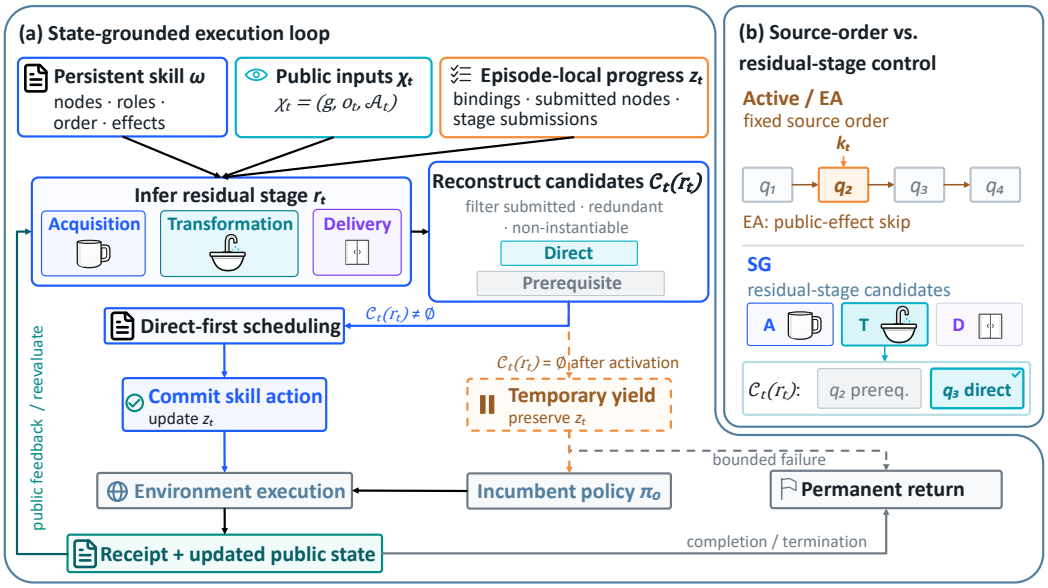


Figure 2: State-grounded execution. SG reconstructs residual-stage candidates from a persistent skill, public inputs, and episode-local progress. It schedules direct actions first and preserves z_t across temporary yields; Active and EA instead advance a monotone source pointer.

episode-local progress z_t, SG binds source roles, infers a residual stage, constructs source-supported candidates in that stage, and either submits a candidate or yields the decision to the incumbent (Figure 2).

4.1 Skill binding and invocation progress

A compiled skill contains m source nodes $\mathcal{O}_\omega = (q_1, \ldots, q_m)$. Each node stores a source index, operation template, operator, variables, and roles such as `object` and `target`. A fixed parser extracts roles from g and matches instantiated nodes against $\mathcal{A}_t$. For selected core operators, a failed template match falls back to the lexicographically first admissible command with the required operator and object name without changing existing bindings. Every skill action is therefore source-supported and publicly admissible.

The transient state z_t records bindings, submitted node indices, visit count, stage submissions, repair waits, and completion/failure flags. It is updated when an action is submitted; the subsequent environment receipt and observation are stored separately. Thus, submitted progress records controller commitment, not semantic verification of an effect.

4.2 Residual-stage scheduling

Two public scheduling proxies connect observations to progress: $\mathrm{ObsEffect}(\hat{s}_t)$ detects a registered effect in the current observation, and $\mathrm{PublicHold}(\mathcal{A}_t)$ detects commands that expose moving the target object. The scheduling variables combine these public proxies with submitted progress:

$$\begin{aligned} \mathrm{sat}_t^{\mathrm{sched}} &= z_t.\texttt{effect_done} \vee \mathrm{ObsEffect}(\hat{s}_t), \\ \mathrm{hold}_t^{\mathrm{sched}} &= z_t.\texttt{option_holding} \vee \mathrm{PublicHold}(\mathcal{A}_t). \end{aligned} \tag{5}$$

Here, n_t^{del} counts target-delivery actions submitted by this invocation and n^{req} is the number of deliveries requested by the goal. For an active, nonterminal invocation, the following cases are evaluated in order for transformation operator e_ω:

$$r_t = \begin{cases} \text{acquisition}, & \neg\,\mathrm{hold}_t^{\mathrm{sched}} \wedge n_t^{\mathrm{del}} < n^{\mathrm{req}}, \\ \text{transformation}, & e_\omega \neq \varnothing \wedge \neg\,\mathrm{sat}_t^{\mathrm{sched}}, \\ \text{delivery}, & \text{otherwise}. \end{cases} \tag{6}$$

These proxies and counts control scheduling; full-task success is evaluated independently by the environment verifier.

Within r_t, SG removes submitted, redundant, and non-instantiable nodes and classifies the remainder as direct stage actions or supporting prerequisites. Let $\mathcal{C}_t(r_t)$ be this candidate set and let $p_{r_t}(j) = 0$ for direct actions and 1 for prerequisites. The frozen scheduler chooses

$$j_t^{\mathrm{SG}} = \arg \min_{j \in \mathcal{C}_t(r_t)} \big(p_{r_t}(j), \mathrm{sourceIndex}(j), \mathrm{commandString}(j)\big). \tag{7}$$

It therefore prefers direct progress in the residual stage, retains source order within a priority class, and breaks remaining ties deterministically. An admissible delivery command cannot jump ahead of an unfinished acquisition or transformation stage. This reconstruction avoids a source-prefix failure: when the pointer's current node is no longer instantiable but a later direct stage action is admissible, SG can issue that action without replaying the prefix. The submitted-node set prevents repeated issuance, while temporary yield lets the incumbent expose a missing prerequisite without discarding invocation progress.

4.3 Bounded control over the incumbent

Committing node j_t records its source index, successful template bindings, and the corresponding stage submission before environment execution. A target-delivery submission advances the registered delivery count; reaching its threshold marks controller completion. The environment independently returns a receipt and next observation.

If an active skill has no candidate, SG preserves z_t and temporarily yields to the incumbent. A later public state can expose a candidate and resume the same activation. After activation, six consecutive scheduler calls with no candidate mark the skill as failed; a committed skill action resets this counter, and option visits are bounded by $2m$. Completion, failure, or environment termination permanently returns control and prevents a second activation in the episode.

Activation uses a deterministic as-applicable caller. A side-effect-free preview first checks for a candidate; activation then charges and discards one incumbent draft for the replaced decision. Malformed or inadmissible output consumes calls and tokens but no environment action. Appendix A specifies the complete event contract.

4.4 Algorithmic difference

The comparison executors are also stateful. Active and EA retain a monotonically advancing source pointer and bindings; EA additionally skips the current transformation when its public effect is satisfied. SG instead reconstructs candidates across all unsubmitted nodes in the residual stage, applies direct-first scheduling, and preserves the invocation across temporary yields. Active, EA, and SG share the outer runner, public binding interface, fallback permissions, and initial budgets; their candidate selection and no-command termination rules differ as specified in Appendix A.

5 Experimental Setup

We isolate execution control after fixing the receiver, procedural knowledge, public information, permissions, and budgets. The primary study compares SG with EA on prospectively selected exact tasks under matched assets and interfaces. A complete official-split study measures distribution-level utility, and a secondary study compares SG with Active on the same official manifests. These studies retain separate task and statistical identities; Appendix C provides the full protocol map.

Environment and receiver. We use the text interface of ALFWorld (Shridhar et al., 2021), which exposes observations and admissible commands for household tasks. Full-policy evaluations use Llama-3.1-8B-Instruct with BF16 weights, SDPA attention, and greedy decoding. The prompt, wrapper, parser, feedback format, and model parameters are fixed within each comparison. Success comes from the original environment verifier.

The evaluated portfolio contains cleaning and placement skills. Cleaning uses a fixed retrieval-augmented incumbent; placement uses a receiver-matched LoRA incumbent. Within each scope,

all executors share the same incumbent and compiled skill. The eight-node cleaning skill spans prerequisites, acquisition, cleaning, and delivery; the four-node placement skill spans acquisition and delivery. Target object and destination roles use the shared public binding interface.

Tasks and policies. The official study evaluates all 140 `valid_seen` and 134 `valid_unseen` tasks; unsupported scopes remain in the denominator and continue under the incumbent. The `valid_unseen` split was evaluated only after the portfolio, method, and budgets were frozen, and intervals use lineage-clustered resampling.

The primary comparison uses 120 prospectively selected train tasks balanced across clean and place. They have zero exact-task overlap with prior experiments and cover 109 lineages; 90 share a historical lineage, making the design exact-task prospective rather than lineage-held-out. Four policies run independently from reset. **Retain** uses the scope-specific incumbent. **Active** is a stateful source-order executor. **EA** adds public effect skipping and is the primary comparator. **SG** reconstructs candidates from submitted progress and residual stage. All three option executors share the runner and binding interface, but Active and EA fail when their current source rule cannot instantiate after activation, whereas SG can yield with z_t preserved.

Budgets, endpoints, and statistics. Each episode permits at most 40 model calls and 40 environment actions, 192 new tokens per call, 6,144 output tokens total, and an 8,192-token context. Each policy–task pair contributes one rollout, with failures and budget exhaustion retained in the denominator. A discarded incumbent draft is charged on option activation.

The primary endpoint is full-task success. For policies π and π_c, the paired difference is

$$\widehat{\Delta S}_{\pi|\pi_c} = \frac{1}{n}\sum_i (Y_i^{\pi} - Y_i^{\pi_c}) = \frac{n_{10} - n_{01}}{n}. \tag{8}$$

We use a two-sided exact McNemar test and 20,000 paired exact-task bootstrap resamples. The joint criterion requires a positive success difference and a priced-runtime interval whose upper bound is at most zero. Official and Active intervals cluster by lineage; the timing study clusters repetitions by task identity. Sample-size design and estimator details appear in Appendix C.

6 Results

6.1 Execution changes full-task outcomes under a fixed skill bank

On the 120-task matched-skill comparison, SG solves 56 tasks versus 36 for EA. All 20 discordant pairs favor SG, giving a success improvement of 16.67 percentage points (95% paired-bootstrap CI: $[10.00, 23.33]$; two-sided exact McNemar $p = 1.91 \times 10^{-6}$). This improvement accompanies 23.16% lower priced runtime (paired difference -0.033980, 95% CI $[-0.047812, -0.021381]$) and 23.18% lower wall-clock time. Model calls, environment actions, and token use also decrease. Figure 3 summarizes the joint success–resource result; Table 8 gives the full paired ledger.

6.2 The fixed portfolio improves complete official splits

The official-split evaluation includes task types unsupported by the portfolio. SG solves 39/140 `valid_seen` tasks versus 22/140 for Retain, and 41/134 `valid_unseen` tasks versus 28/134. The lineage-clustered improvements are 12.14 and 9.70 percentage points, respectively (Table 1).

Active retains fixed source order while sharing SG's runner, binding, fallback permissions, and budgets. SG improves complete-split success over Active by 7.14 points on seen and 6.72 points on unseen; all 19 discordant tasks favor SG. A separate 24-task exclusive-device crossover also favors SG in success and wall time; Appendix D reports the full Active, Ordinary, timing, and resource results.

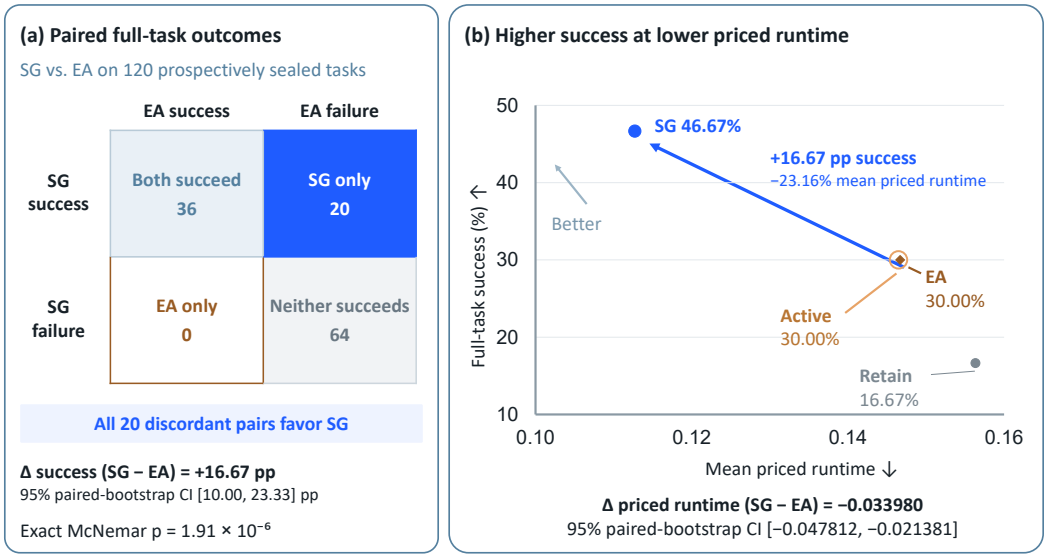


Figure 3: Primary matched-skill results. (a) All 20 discordant full-task outcomes favor SG. (b) SG attains the highest success with the lowest mean priced runtime among the four policies.

Table 1: Complete official-split comparisons. Unsupported task types remain in the denominator; intervals cluster by lineage.

Comparator	Split	SG	Baseline	Difference	95% CI
Retain	Seen	39/140	22/140	+12.14 pp	[7.09, 17.86]
Retain	Unseen	41/134	28/134	+9.70 pp	[3.76, 16.79]
Active	Seen	39/140	29/140	+7.14 pp	[2.92, 11.59]
Active	Unseen	41/134	32/134	+6.72 pp	[2.21, 12.59]

7 Analysis and Discussion

7.1 Effect awareness does not determine the residual stage

EA retains a monotone source pointer and skips the current transformation when its public effect is satisfied. In finite-state tests, this branch changes actions when its predicate changes, but Active and EA produce identical traces on the 120 natural tasks. By contrast, replacing residual scheduling with fixed source order in the 140-task structural ablation reduces success by 7.14 percentage points and increases runtime cost. The observed gain therefore concerns the residual controller rather than effect filtering alone.

7.2 Transformation stages locate the main benefit

Table 2A shows the structural concentration. On 60 cleaning tasks, SG solves 30 and EA 11, with 19/0 discordant pairs. On 60 placement tasks, which lack a separate transformation stage, the methods solve 26 and 25. Thus, 19 of 20 SG-only successes arise in the transformation-bearing scope. Clean and place also differ in template, incumbent, and binding structure, so the split locates a benefit region rather than isolating stage count alone.

Table 2: Primary paired outcomes by skill scope.

A. Scope-stratified paired outcomes

Scope	SG / EA	Discordant	Success difference	Runtime-cost difference
Clean ($n = 60$)	30 / 11	19 / 0	+31.67 pp [20.00, 43.33]	-0.063914 $[-0.088569, -0.040778]$
Place ($n = 60$)	26 / 25	1 / 0	+1.67 pp [0.00, 5.00]	-0.004045 $[-0.007987, -0.001439]$

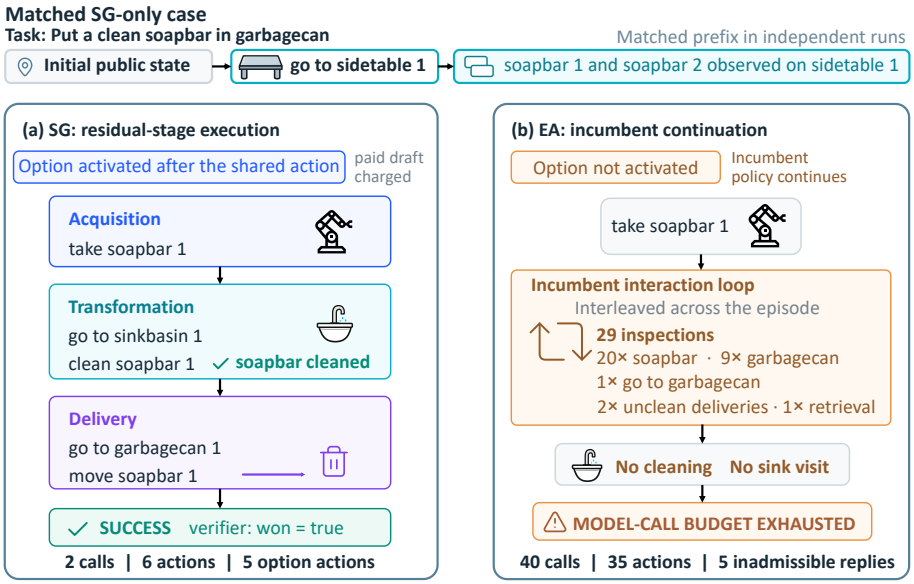


Figure 4: A matched SG-only cleaning case. Independent runs share the initial prefix; SG then activates and completes acquisition, transformation, and delivery, while EA leaves control with the incumbent and exhausts its call budget. The EA loop counts summarize the episode.

7.3 More skill execution, fewer total resources

SG submits 350 option actions versus 189 for EA, yet requires fewer model calls and environment actions overall. Together with higher full-task success, this pattern is consistent with residual execution replacing unproductive interaction with useful skill actions.

Figure 4 shows the corresponding entry difference. After the shared navigation action, EA remains at a source-prefix navigation node that cannot instantiate in the new public state, so its option does not activate. SG reconstructs the acquisition stage and exposes the later admissible `take soapbar 1` node. The case illustrates activation through residual-stage reconstruction. Appendix Figure 5 gives the complete SG action-level trace.

7.4 Limitations

The evidence covers deterministic execution on ALFWorld's public text interface with a fixed clean/place portfolio and one Llama receiver. Resource comparisons measure complete-episode serving with an already-owned skill bank. Except for the repeated exclusive-device timing study, each policy–task pair contributes one greedy rollout, so the estimates characterize this deterministic execution setting.

8 CONCLUSION

SG controls a stored procedure's remaining execution with stage-specific candidates and episode-local progress. Under matched skills and budgets, direct-first scheduling and bounded yield improve full-task success while reducing execution resources. Prospective SG–EA and official-split results show that reuse depends on how the remaining operations are organized at runtime.

AI USE STATEMENT

Generative AI tools were used for grammar and spelling checks, language polishing, related-work retrieval, and programming assistance for a subset of the experiments. All AI-assisted work was reviewed by the authors, who take responsibility for the final content of this work.

ETHICS STATEMENT

This work evaluates language-agent control on the public ALFWorld benchmark and does not involve human subjects, private data, or real-world actuation.

REPRODUCIBILITY STATEMENT

Sections 4–5 and Appendix A specify the executor, comparison rules, and evaluation protocol. The anonymized supplement provides executor semantics, sanitized skill and task specifications, task-level outcomes, selected public-state traces, and scripts for integrity checks, result recomputation, and semantic tests. Full model-and-environment replay additionally requires the separately obtained model, benchmark, and adapter dependencies described in the supplement.

REFERENCES

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

BehaviorTree.CPP. Sequence nodes: Sequence, reactivesequence, and sequencewithmemory. BehaviorTree.CPP documentation, version 4.8, 2026. URL `https://www.behaviortree.dev/docs/nodes-library/SequenceNode/`.

Michele Colledanchise, Diogo Almeida, and Petter Ögren. Towards blended reactive planning and acting using behavior trees. In *IEEE International Conference on Robotics and Automation*, pp. 8839–8845, 2019.

Xueping Gao. Compositional skill routing for LLM agents: Decompose, retrieve, and compose. *arXiv preprint arXiv:2606.18051*, 2026.

Nicholas Hay, Stuart Russell, David Tolpin, and Solomon Eyal Shimony. Selecting computations: Theory and applications. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, pp. 346–355, 2012.

Qirui Mi, Zhijian Ma, Mengyue Yang, Haoxuan Li, Yisen Wang, Haifeng Zhang, and Jun Wang. Skill-Pro: Learning reusable skills from experience via non-parametric PPO for LLM agents. In *International Conference on Machine Learning*, 2026. arXiv:2602.01869v3.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *Proceedings of the International Conference on Machine Learning*, pp. 9229–9248, 2020.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999.

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

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent Workflow Memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025. URL https://proceedings.mlr.press/v267/wang25bx.html.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, Feiyu Xiong, Yutao Qi, Bo Tang, and Muning Wen. MemRL: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*, 2026.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. SkillWeaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

A Complete Controller Specification

A.1 Inputs, outputs, and state

At decision attempt t, the controller receives skill ω, goal g, current observation o_t, admissible commands $\mathcal{A}_t$, episode-local state z_t, and shared budget b_t. It returns an admissible environment action, a temporary yield with z_t preserved, or a permanent return. It reads only the public text interface; Table 3 lists its cursor fields.

Table 3: Episode-local execution state. Commit fields record actions issued by the controller; they are not environment truth.

Field	Meaning	Update / control use
`roles`, γ_t	Target object/destination roles and public instance bindings	Initialized from goal; extended only by successful template matches
`executed_indices`	Source nodes submitted in this activation	Insert node index on commit; filter repeat candidates
`effect_done`	Target transformation submitted	Set on transformation commit; scheduling proxy only
`option_holding`	Acquisition submitted without later target delivery	Set on target take; cleared on target move
`desired/completed_placements`	Required and submitted target deliveries	Completion threshold; two-object goals require two
`visits`	Submitted skill-action count	Maximum $2m$ for an m-node skill
`repair_waits`	Consecutive scheduler calls with no candidate after activation	Reset after skill commit; sixth wait fails
`started/completed/failed`	Activation lifecycle flags	Gate resume and permanent return

A.2 Stage and candidate mapping

Table 4 expands the residual-stage mapping used in Eq. (6). Internal code labels `pickup`, `effect`, and `handoff` correspond to acquisition, transformation, and delivery; the last code label denotes object delivery, not control return.

Table 4: Residual stages and source-supported candidates.

Stage	Scheduling condition	Direct candidates	Supporting candidates
Acquisition	Object not held for scheduling; delivery incomplete	Target-object `take`	Linked `go`/`open`
Transformation	Scope has `clean`, `heat`, or `cool`; effect not satisfied	Registered transform on target	Linked navigation/tool prerequisites
Delivery	Transformation submitted or publicly satisfied; delivery incomplete	Target-object `move` to destination	Destination-linked `go`/`open`
Light special case	Registered effect is `use`	First instantiable source-order node	Source order, without three-stage reclassification

Direct acquisition, transformation, and delivery nodes are identified by operator and role: target-object `take`, the scope's registered transformation, and target-role `move`, respectively. Remaining nodes are assigned by the first matching rule: a target role or variable makes a delivery support node; otherwise variables shared with a transformation node make transformation support; otherwise variables shared with target acquisition make acquisition support. Other nodes are excluded from the three-stage branch. Candidates are removed if already submitted, non-instantiable, or redundant with a satisfied transformation. Direct candidates precede supporting candidates; source index and command string break ties. Placement has no transformation branch, while `light`/`use` follows the source-order special case. A side-effect-free preview applies the same candidate logic without updating state.

The public effect proxy searches the observation text before the appended goal instruction for fixed scope-specific phrases referring to the target object. The holding proxy is true when the current admissible-command set contains a target-object `move` command. Together with `effect_done` and `option_holding`, these proxies produce Eq. (5); they do not inspect latent state or the task verifier.

A.3 Event and budget contract

Table 5 makes explicit when state and budgets change. The runner stores the full public event record. The incumbent prompt is limited to the goal, the most recent 12 public transitions, current observation, admissible commands, and error feedback.

The environment-action limit governs the outer episode loop. Model-call and output-token limits are checked before each generation. Consequently, a previously activated skill may issue a no-generation action after the final allowed model call, but the episode stops when another generation is required or another registered termination condition holds.

A.4 Executor mappings

Table 6 records the implementations used to obtain the reported rows. All four executors keep episode-local state; their difference is how that state determines the next source-supported action.

Ordinary, Active, and EA mark controller completion when the source pointer reaches the end of the stored templates. SG marks controller completion when submitted target-delivery commands reach the goal-derived delivery count. These are controller events; every full-task success in the paper comes from the independent environment verifier. Under bounded template and command lengths and at most b active bindings, constructing candidate binding copies yields $O(L_t + m(a \log a + b) + m \log m)$ time and $O(mb + a)$ auxiliary workspace per SG decision. For fixed b, the time expression reduces to the corresponding constant-binding bound.

Table 5: Execution-event contract. A decision round need not contain an environment action.

Event	Controller / runner effect	Charged resources	Public trace / effect
Entry preview	None	None	None
Paid draft	Runner marks the opportunity used; cursor remains unstarted until its first commit	One model call and generated tokens	Draft discarded; no history mutation
Template commit	Insert source node, merge matched bindings, update visit/stage fields	Scheduling overhead; no model call	Command sent to public interface
Operator fallback commit	Insert source node; preserve existing bindings	Scheduling overhead; no model call	Lexicographically first matching admissible command sent
Environment receipt	No reinterpretation of commit fields	One environment action and measured runtime	Append action, receipt, and next observation
No-candidate after start	Increment repair wait; preserve state before sixth wait	Incumbent macro or generation if needed	Temporary yield; action only if incumbent output is valid
Malformed/ inadmissible reply	No environment progress	Model call and tokens charged	Record rejection; no environment action
Completion/ failure	Set terminal cursor state	Already incurred resources remain	Permanent return; no reactivation this episode

Table 6: Executor state transitions. "No command" means that the current rule cannot instantiate to a public admissible command.

Executor	Progress and advancement	Effect skip, no-command behavior, and fallback
Ordinary	Source pointer and bindings; pointer advances on command commit	Public effect skip advances past the transformation; no command before entry prevents activation and after entry fails; no operator fallback
Active	Source pointer, symmetric role bindings, and visits; pointer advances on commit	No effect skip; no command before entry prevents activation and after entry fails; shared goal-object fallback preserves bindings
EA	Active state plus permanent pointer advance across a publicly satisfied transform	Effect skip only at the current transformation; otherwise Active no-command and fallback rules
SG	Submitted-node set, bindings, stage fields, delivery count, and visits	Reconstructs residual-stage candidates; after activation, empty sets yield and the sixth consecutive empty scheduler call fails; shared core-operator fallback

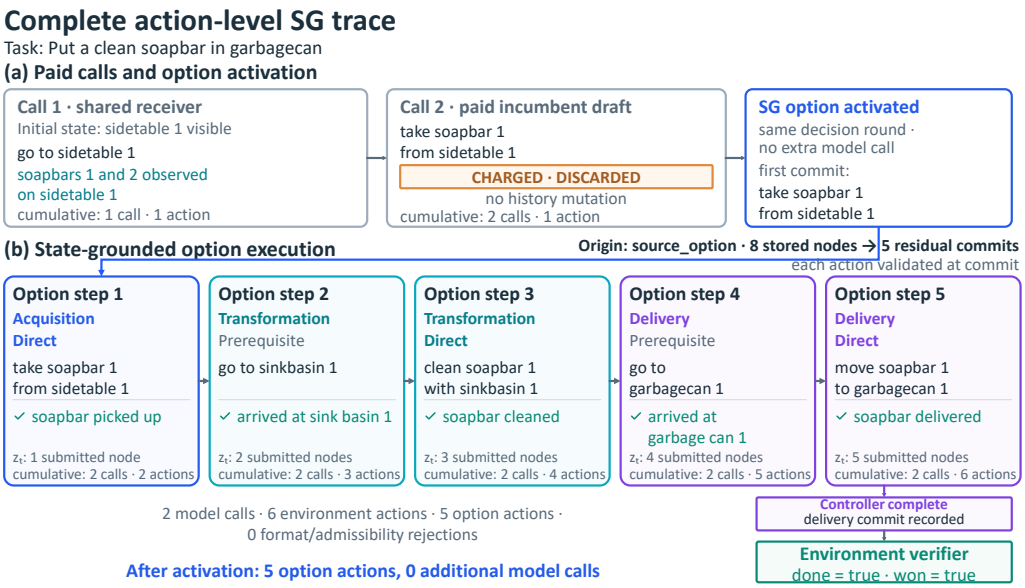


Figure 5: Complete SG action-level trace for the matched case in Figure 4. After the charged incumbent draft is discarded, five residual actions complete acquisition, transformation, and delivery without another model call; controller completion and environment-verified success remain distinct.

B Finite-State Witnesses and Qualitative Traces

B.1 Effect-aware execution is functional

The same cleaning skill was evaluated under two public states. When the observation reported that the mug was clean, Active selected `clean mug 1 with sinkbasin 1`, while EA skipped that node and selected `move mug 1 to cabinet 1`. When cleanliness was unknown, EA retained the clean action. Thus, the effect-completion branch changes behavior when its predicate changes.

B.2 Episode-local progress changes SG actions

With the persistent skill, observation, and admissible commands held fixed, a fresh SG cursor selects the clean action. After that node is recorded as submitted, its next action becomes delivery. This verifies that z_t changes action selection without treating submitted progress as environment truth.

B.3 Selected public traces

Figure 5 expands the successful SG branch of the matched case in Figure 4. The anonymized supplement includes the corresponding public action–observation records. These records retain admissible commands, committed actions, receipts, submitted progress, termination status, and resource accounting while excluding model free-text responses and simulator-private state.

C Statistical Protocol and Evidence Identity

For paired success, let n_{10} and n_{01} denote SG-only and comparator-only tasks. The two-sided exact McNemar value is

$$p = 2\Pr[\mathrm{Binom}(n_{10}+n_{01}, 0.5) \le \min(n_{10}, n_{01})]\,, \tag{9}$$

truncated at one. The primary comparison has $n_{10} = 20, n_{01} = 0$, yielding $p = 1.9073486328125 \times 10^{-6}$.

Table 7: Identity and role of the studies contributing paper results. Studies are analyzed separately.

Study	Data identity	Paper role	Statistical unit
Official split	Complete `valid_seen`/ `valid_unseen` manifests	Distribution-level gain over Retain and Ordinary	Lineage cluster
Active secondary	Active comparator on the official manifests; separate timing set	Stateful source-order baseline, bank analysis, and physical timing	Lineage; timing task identity
Source acquisition	Independent exact-task source attempt	Availability of an additional cleaning-capable bank	Fixture, source task, bank
Matched skill	Prospective exact-task-nonoverlapping tasks on the owned bank	Primary SG–EA success/resource test	Exact task

All matched-skill differences use 20,000 paired exact-task bootstrap resamples. Official results condition on the fixed portfolio and use task-weighted lineage-clustered bootstrap intervals: 137 groups for `valid_seen` and 47 for `valid_unseen`.

D Extended Results

D.1 Caller and scheduling attribution

In a caller study, an incumbent-relative tree, a strong-simple rule, and the final deterministic caller made identical decisions on all 40 legal intervention opportunities. A separate 140-task structural ablation replaced residual scheduling with fixed source order; success decreased by 7.14 percentage points and runtime cost increased. The first comparison fixes the caller choice; the second isolates the execution scheduler.

D.2 Primary comparison and source strata

Table 8: Extended paired results on the 120-task prospective matched-skill set.

Metric / subset	SG / comparator	Difference / discordance	95% paired interval
Full success vs. EA	56/120 / 36/120	+16.67 pp; 20/0	$[10.00, 23.33]$ pp
Priced runtime vs. EA	0.112716 / 0.146695	-0.033980	$[-0.047812, -0.021381]$
Success vs. Retain	56/120 / 20/120	+30.00 pp; 37/1	$[21.67, 38.33]$ pp
Clean scope	30/60 / 11/60 (EA)	+31.67 pp; 19/0	$[20.00, 43.33]$ pp
Place scope	26/60 / 25/60 (EA)	+1.67 pp; 1/0	$[0.00, 5.00]$ pp

Policy	Input tokens	Output tokens	Wall time (s)
Retain	55,048.3	533.1	15.612
Active source-order	47,307.4	483.5	14.094
EA	47,307.4	483.5	14.110
SG	34,852.0	359.9	10.839

Table 9: Extended results from the official and Active-secondary studies. Each panel preserves its own unit of analysis.

Setting	SG / baseline	Difference	95% interval
Official seen vs. Retain	39/140 / 22/140	+12.14 pp	$[7.09, 17.86]$
Official unseen vs. Retain	41/134 / 28/134	+9.70 pp	$[3.76, 16.79]$
Official seen vs. Ordinary	39/140 / 25/140	+10.00 pp	$[5.48, 15.11]$
Official unseen vs. Ordinary	41/134 / 28/134	+9.70 pp	$[3.76, 16.79]$
Official seen vs. Active	39/140 / 29/140	+7.14 pp	$[2.92, 11.59]$
Official unseen vs. Active	41/134 / 32/134	+6.72 pp	$[2.21, 12.59]$
Active seen candidate scopes	34/62 / 24/62	+16.13 pp	$[7.94, 25.81]$
Active unseen candidate scopes	26/55 / 17/55	+16.36 pp	$[5.45, 29.63]$
Timing success (72 blocks)	36/72 / 21/72	+20.83 pp	$[4.17, 37.50]$
Timing mean wall time/task	11.269 s / 15.052 s	-3.783 s	$[-7.105, -1.118]$
Timing priced runtime/task	–	-0.03646	$[-0.07142, -0.00666]$

Bank	Supported scopes	SG / Active	Success difference
Bank 00	Place	12/24 / 14/24	-8.33 pp
Bank 01	Place	16/24 / 16/24	0.00 pp
Bank 02	Clean + place	26/55 / 17/55	+16.36 pp

D.3 Official, active, bank, and timing results

Bank is the construction-replication unit: the pooled 103 bank–task pairs are not 103 independent banks. The bank supporting cleaning reproduces the large cleaning gain, while the three placement-capable banks show success-count differences of $-2, 0, 0$.

D.4 Source-semantic interventions

Table 10: Source-semantic interventions on skill content.

Condition	Pairs	SG success	Control success	SG-only / control-only
Public-only source	28	16	14	6 / 4
Semantic deletion	28	16	12	5 / 1
Compatible replacement	28	16	14	4 / 2
No-source relations	140	39	39	15 / 15

E Reproducibility

The anonymized artifact provides sanitized skill and task specifications, task-level outcomes, selected public traces, analysis code, and executor tests for result recomputation. Full model-and-environment replay requires separately obtained ALFWorld assets, model weights, and adapters.