PROCEDURAL EDITING: CLOSED-LOOP REPAIR OF SOP VIOLATIONS IN TOOL-USE LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Standard Operating Procedures (SOPs) govern tool-use workflows by specifying which actions are permitted, under what conditions, and in what order. A single skipped check, premature tool call, or early termination can therefore invalidate an otherwise plausible outcome. Such failures are natural targets for model editing: an SOP checker can localize the first divergent decision and provide the context, emitted action, and required replacement. However, we identify two reasons why traditional model editing transfers poorly to agent trajectories. First, reasoning agents act from a decision state that includes their generated rationale; consequently, an edit can achieve perfect teacher-forced rewrite accuracy yet leave the live action unchanged. Second, correcting one decision rarely repairs an entire trajectory, because the agent may diverge again downstream. We formulate *procedural editing* as closed-loop, trajectory-level repair and introduce PACE (Procedure-Aware Closed-loop Editing). PACE optimizes corrections at the recorded decision state, reruns the agent after every update, converts each new divergence into the next edit, and strengthens a correction when execution returns to the identical failing state. Across 119 SOPBench incidents, two Qwen3 models, and both scaffolds, PACE achieves the highest or tied-highest SOP success in all four settings, reaching 0.50–0.51 under ReAct and 0.52–0.66 under function calling. It retains 0.81–0.86 of previously passing tasks, while preserving general-capability.[1]

1 INTRODUCTION

LLM agents increasingly execute long-horizon workflows in customer support, financial verification, medical intake, and other procedure-governed settings. Correctness in these settings is not determined by the final answer alone. *Standard Operating Procedures* (SOPs) specify which actions are required or permitted, the conditions under which they may occur, and their order. An agent that issues a refund before checking eligibility or approves a KYC (Know Your Customer) case without mandatory verification has failed even if it eventually reaches a plausible outcome. ReAct and function-calling agents exhibit such permissibility, ordering, and completeness violations on SOP-guided tasks (Nandi et al., 2025; Li et al., 2025). Importantly, every failed trajectory has an earliest divergent decision: a context in which the agent emits the wrong tool, arguments, action, or stopping decision.

That structure makes SOP failures a natural target for model editing. A checker-localized incident already provides the ingredients of a correction: the context before the violation, the action the agent emitted, and an SOP-valid replacement, all without requiring a separate training corpus. A localized weight edit can then change the model's action preference while a projected update protects unrelated behavior (Fang et al., 2025). This approach complements runtime enforcement through decision graphs, tool filtering, or constrained generation (Ye et al., 2025; Koo et al., 2024): enforcement blocks invalid behavior at execution time, whereas procedural editing attempts to repair the policy that produced it.

However, factual model editing is built around a static prompt–target association, and two assumptions behind that setting break for agent trajectories. First, *scoring-site/sampling-site identity* (A1): factual targets immediately follow the prompt, but a ReAct agent generates a rationale before acting.

[1] Code and data are available at https://anonymous.4open.science/r/procedural-edit

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.

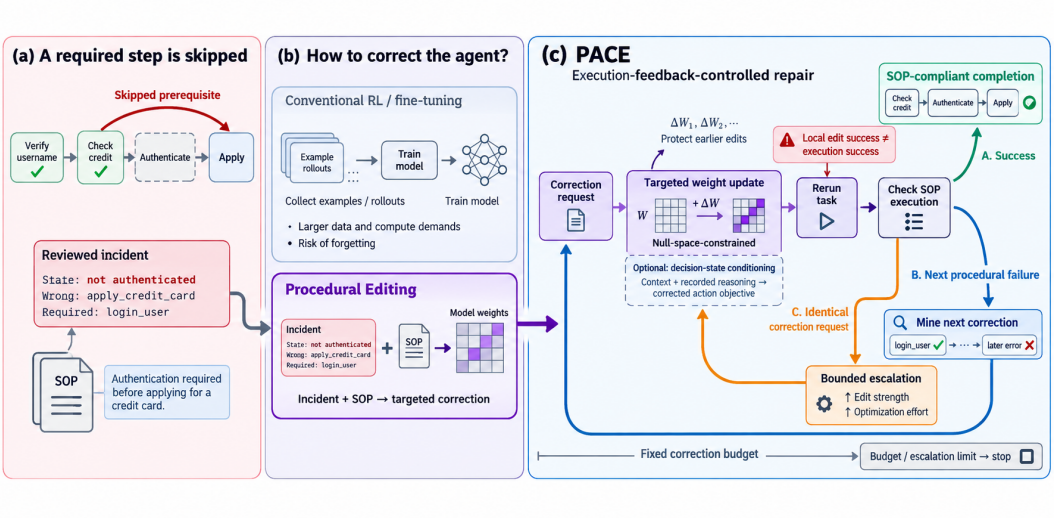

Figure 1: PACE repairs SOP-following failures through targeted model edits and closed-loop execution feedback. Each rollout is checked against the SOP; the first divergence becomes an edit request, the edit is installed with a null-space-projected solve, and the edited agent is rerun. A new divergence starts a new edit; an identical divergence escalates the existing edit.

Its deployed action is therefore sampled from a decision state that includes the generated rationale, not merely from the context at which a conventional editing method scores the target. In our experiments, 48–77% of first edits that achieve perfect teacher-forced rewrite accuracy under ReAct still fail to emit the corrected action in the live rollout; the corresponding pooled rate under native function calling is only 1–6%. We call these edits *installed but inert*. Second, *single-decision sufficiency* (A2): correcting the first violation does not guarantee that the remainder of a long-horizon trajectory follows the SOP. Across our settings, one PACE edit repairs only 0.27–0.31 of failing trajectories, whereas up to five feedback-selected edits raise success to 0.50–0.66.

We formulate *procedural editing* as closed-loop, trajectory-level repair. After each parameter update, the agent is rerun from the task start and the SOP checker evaluates the complete trajectory. Success terminates the loop; a later divergence supplies the next correction. We apply this protocol to every editing and training baseline, separating the benefit of execution feedback from the mechanism used to install each correction.

Within this protocol, we introduce PACE (Procedure-Aware Closed-loop Editing; Figure 1). PACE adds two mechanisms to a null-space-projected locate-and-edit backend. *Decision-state value conditioning* optimizes the replacement action at the recorded state from which a reasoning agent actually acted, including its pre-action rationale. *Bounded no-progress escalation* detects a byte-identical return to the same divergence and strengthens that correction along a finite configuration ladder, rather than repeating an unchanged update. A new divergence remains new supervision. Thus, the live trajectory determines both what to edit next and whether an installed edit was behaviorally effective.

Our contributions are:

1. We formulate procedural editing as sequential, checker-supervised repair evaluated by trajectory-level SOP success. We identify and measure two failures of factual editing in this setting: *installed-but-inert* edits caused by a scoring-site/sampling-site mismatch, and the insufficiency of repairing only one decision in a long-horizon trajectory.
2. We introduce PACE, which combines decision-state value conditioning and bounded no-progress escalation with a diagnose–install–verify–adapt controller. It preserves the localization and projected solve of its editing backend while changing the state at which a target value is optimized and the execution conditions under which a correction is strengthened.
3. We run comprehensive experiment cross 119 incidents, 2,499 repair loops, and 62,370 held-out rollouts, PACE achieves the highest or tied-highest mean SOP success in all four model–scaffold settings, reaching 0.50–0.66 with five edits and significantly outperforming AlphaEdit in three settings. It retains 0.81 of previously passing own-domain tasks under ReAct and 0.86 under function calling, changes general-capability accuracy by at most 0.09 percentage points, and installs edits about 3× faster than GRPO under function calling.

2 Related Work

Model editing and procedural agents. ROME and MEMIT edit factual associations in feed-forward layers (Meng et al., 2022; 2023); AlphaEdit projects updates onto the null space of preserved keys to protect unrelated associations across sequential edits (Fang et al., 2025). Recent work extends model editing from factual knowledge to agent behavior (Wang et al., 2024a; Zhang et al., 2024; Huang et al., 2026). In parallel, ReAct interleaves reasoning and action (Yao et al., 2023), while decision graphs and constrained generation enforce procedures externally (Ye et al., 2025; Koo et al., 2024; Beurer-Kellner et al., 2024). SOPBench, SOP-Bench-amz, τ-bench, and τ^2-bench evaluate policy compliance in tool-using agents (Li et al., 2025; Nandi et al., 2025; Yao et al., 2024; Barres et al., 2025). PACE instead edits the model's action preference at a divergence discovered through execution and evaluates the result by live trajectory-level success.

Learning from failures. Reflexion, ExpeL, and Agent Workflow Memory retain verbal experience in context (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2024b); LoRA and GRPO optimize parameters from trajectories (Hu et al., 2022; Shao et al., 2024); and process supervision localizes incorrect reasoning steps (Lightman et al., 2024). PACE writes one checker-localized correction into the weights, where it persists across context resets, and uses subsequent execution to select the next correction.

3 Problem Formulation

Consider an agent comprising a parameterized model f_θ and a tool-use scaffold g (e.g., ReAct or function calling) that interact with an environment $\mathcal{M}$. On a task x, their interaction produces a trajectory $\xi = \mathrm{Rollout}(f_\theta, g, \mathcal{M}, x)$. An SOP checker $\mathcal{V}$ returns either SUCCESS or the earliest *procedural divergence*

$$d = (c, a^-, a^*), \tag{1}$$

where c is the deploy context immediately before the error, a^- is the action emitted by the agent, and a^* is the action required by the SOP. For weight editing, we augment d with an anchor s for key extraction and, when present, the model's recorded pre-action reasoning $\hat{\tau}$, giving the edit request

$$e = (c, s, a^-, a^*, \hat{\tau}). \tag{2}$$

The deploy context c is the rendered transcript from the system prompt through the last observation before the divergence; the anchor s is a subject-like token span, such as the account username, whose last occurrence in c provides the key-extraction site. The *decision state* is the state from which the agent emits its action: c itself when the action directly follows the context, as under function calling, and $c \oplus \tau$ when the agent first emits reasoning τ, whose recorded instance is $c \oplus \hat{\tau}$. In our experiments, Qwen3 runs with thinking mode disabled, so τ is the ReAct `Thought` text. Unlike a factual tuple supplied independently of model behavior, this request is discovered by executing the policy: one failed trajectory supplies one supervised correction.

Procedural repair is sequential. Starting from θ_0 on a failing task x, an editing operator $\mathcal{U}$ installs the correction mined from the checker's divergence, and the edited agent is executed again:

$$\xi_t = \mathrm{Rollout}(f_{\theta_t}, g, \mathcal{M}, x), \tag{3}$$

$$d_t = \mathcal{V}(\xi_t), \tag{4}$$

$$\theta_{t+1} = \mathcal{U}(\theta_t, e_t; \kappa_t), \tag{5}$$

where e_t is the edit request mined from d_t and κ_t denotes the edit configuration selected from execution feedback (Algorithm 1 expands $\mathcal{U}$ into value optimization and the projected solve). The process stops at the first T with $\mathcal{V}(\xi_T) = \text{SUCCESS}$, when no valid correction can be mined, or when it exhausts a budget of B installations. The objective is trajectory-level SOP success over a distribution $\mathcal{D}$ of failing tasks,

$$\max\ \mathbb{E}_{x\sim\mathcal{D}}[\mathbf{1}\{\mathcal{V}(\mathrm{Rollout}(f_{\theta_T}, g, \mathcal{M}, x)) = \text{SUCCESS}\}] \quad \text{s.t.} \quad T \le B, \tag{6}$$

while preserving behavior outside the edited states, which we measure by locality and general knowledge and reasoning capability (Refer to Section 5). The objective evaluates trajectory-level execution rather than $p_{\theta_{t+1}}(a^* \mid c)$ on a teacher-forced probe. This formulation separates the *controller*, which decides what to repair next and how to spend the budget, from the *editing operator*, which realizes an individual parameter update.

4 METHODOLOGY

PACE (Procedure-Aware Closed-loop Editing) realizes the controller as a four-stage loop (Algorithm 1): DIAGNOSE the first SOP divergence, INSTALL a localized correction, VERIFY it by rerunning the live agent, and ADAPT the next update to the observed trajectory. A successful rollout terminates the loop; a later divergence becomes a new edit; and a byte-identical divergence triggers bounded no-progress escalation. A conventional editing method treats successful insertion of a^* on a static probe as completion; PACE treats only SOP-compliant execution as completion.

We apply the closed-loop protocol of Section 3 to every method, so all baselines also receive feedback-selected follow-up edits. PACE adds two mechanisms on top of this protocol: decision-state value conditioning, which addresses installed-but-inert edits (A1), and bounded no-progress escalation, which lets the loop make progress when a correction stalls (A2). PACE requires a backend that localizes a request and optimizes a target value; it controls the state at which that value is optimized and the conditions under which the backend is invoked again. We instantiate the backend with AlphaEdit's null-space-constrained update (Fang et al., 2025), a strong locality-preserving editing method. Disabling both mechanisms recovers closed-loop AlphaEdit exactly.

Locate-and-edit model editing methods treat a transformer's feed-forward down-projection as an associative memory mapping a key k to a value v. AlphaEdit constrains updates to the approximate null space P of preserved keys and protects earlier edit keys K_p; Appendix C gives the full objectives and derivations.

4.1 WHY FACTUAL-EDITING ASSUMPTIONS FAIL ON PROCEDURAL EDITS

(A1) Scoring-site/sampling-site identity. For facts, the target tokens immediately follow the prompt, so optimizing $p_\theta(a^* \mid c)$ optimizes deployed behavior. A ReAct agent interposes a reasoning span τ before its action: the action is sampled from the hidden state $h(c \oplus \tau)$, not $h(c)$. A value optimized at the context end can pass teacher-forced probes (rewrite accuracy 1) while the live rollout regenerates its reasoning and re-emits the wrong action: the edit is *installed but inert*. We define the *inert rate* as the fraction of first edits with rewrite accuracy 1 whose live rollout does not emit a^* at the edited decision point. On the edit sets of Section 5.1, the inert rate is 48–77% under ReAct across four locate-and-edit methods (ROME, MEMIT, AlphaEdit, and PACE; 62–63% when pooled per model), versus 1–6% pooled per model under native function calling, where the model emits the tool call with no intervening prose.

(A2) Single-decision sufficiency. For facts, one correct association is the deliverable. Procedurally, correcting one decision does not make a trajectory SOP-compliant: after the correction, the agent proceeds and can diverge again at a later state. This holds across editing methods: after one edit, AlphaEdit repairs 0.14 and 0.29 of ReAct trajectories on Qwen3-4B and Qwen3-8B (Table 5), and PACE repairs 0.27–0.31 across settings; with five edits, PACE reaches 0.50–0.66 (Figure 2).

PACE addresses (A1) with decision-state value conditioning and (A2) with closed-loop repair and bounded no-progress escalation.

4.2 DECISION-STATE VALUE CONDITIONING

PACE computes the target value z^* by inserting the recorded reasoning span $\hat{\tau}$ between the deploy context and the action tokens, changing the objective from $p_\theta(a^* \mid c)$ to $p_\theta(a^* \mid c \oplus \hat{\tau})$. The key remains at the anchor's last token in c. Because appended text cannot causally affect that token, the edit still fires at deploy time. PACE instead changes the value, optimizing it so that its effect on the action survives the model's intervening reasoning. The model's on-policy rationale $\hat{\tau}$ makes $c \oplus \hat{\tau}$ the decision state the edited agent is most likely to revisit. Under function calling, $\hat{\tau}$ is empty and the objective reduces to AlphaEdit's.

Model-level ablations determine whether conditioning is enabled. The experiments enable it for Qwen3-4B, where the conditioned objective uses 40 value-optimization steps rather than AlphaEdit's 25 to converge. For Qwen3-8B, the ablations select escalation alone, making PACE AlphaEdit plus escalation in that setting.

Algorithm 1 PACE: Procedure-Aware Closed-loop Editing

Require: model f_{θ_0}, scaffold g, environment $\mathcal{M}$, failing task x, SOP checker $\mathcal{V}$, backend $\mathcal{U}$, budget B, ladder $\kappa^{(0)}, \ldots, \kappa^{(J)}$

$\theta \leftarrow \theta_0$; $K_p \leftarrow [\,]$; rung $\leftarrow \{\}$; $b \leftarrow 0$
while $b < B$ **do**
  $\xi \leftarrow \mathrm{Rollout}(f_\theta, g, \mathcal{M}, x)$; $d \leftarrow \mathcal{V}(\xi)$ VERIFY / DIAGNOSE
  if $d =$ SUCCESS **then**
    return θ
  end if
  mine $e = (c, s, a^-, a^*, \hat{\tau})$ from d; **if** no valid e **then return** θ
  $\sigma \leftarrow \mathrm{sig}(c, a^-, a^*)$
  if $\sigma \notin$ rung **then**
    $\mathrm{rung}[\sigma] \leftarrow 0$ new divergence
  else if $\mathrm{rung}[\sigma] < J$ **then**
    $\mathrm{rung}[\sigma] \leftarrow \mathrm{rung}[\sigma] + 1$ no progress: escalate
  else
    return θ ladder exhausted
  end if
  $z^* \leftarrow \arg\max_z p_\theta(a^* \mid c \oplus \hat{\tau}; z)$ under $\kappa^{(\mathrm{rung}[\sigma])}$ ($\hat{\tau}$ empty when conditioning is off)
  $\theta \leftarrow \mathcal{U}(\theta, K_e, z^*; P, K_p)$; append K_e to K_p; $b \leftarrow b + 1$ INSTALL
end while
return θ

4.3 Closed-loop repair and bounded no-progress escalation

PACE rolls out the edited agent, mines the first surviving divergence into a new edit request, installs it under the same projected solve with all previously installed keys added to the protected set K_p, and repeats up to a budget of B edits. The trajectory is both the source of supervision and the verification signal.

PACE canonicalizes each divergence into a signature containing its deploy context, emitted action, and target action. After every rollout, the controller applies three cases: SUCCESS terminates; a previously unseen signature starts a new correction at configuration $\kappa^{(0)}$; and a repeated signature, indicating a rollout that re-diverges to a byte-identical failing state, advances along a finite ladder $\kappa^{(0)}, \ldots, \kappa^{(J)}$ until no rung remains. A later divergence is new supervision; a byte-identical divergence is evidence that the current write is too weak. Repeating the same update unchanged would spend budget without adding information, so each rung raises the write's norm allowance and value-optimization effort, and each rung is attempted at most once per signature. The anchor, extracted key, projector, and solver remain fixed, so escalation strengthens the existing correction rather than issuing a new edit request. In all experiments, the ladder has one escalation rung ($J = 1$), which multiplies the value-norm clamp by 1.5 and adds 15 value-optimization steps.

Earlier corrections are protected during later writes. At step t, the cache

$$K_p^{(t)} = [K_e^{(1)}, \ldots, K_e^{(t-1)}] \tag{7}$$

enters the projected solve, penalizing interference with installed edit keys while P protects the reference-key subspace. The controller consumes one unit of budget only when it invokes the backend, so B bounds cumulative parameter updates rather than environment turns.

4.4 Summary of the update

Per critical layer ℓ, PACE solves AlphaEdit's projected least-squares problem (Appendix C),

$$\Delta^{(\ell)} = R K_e^\top P \left(K_e K_e^\top P + K_p K_p^\top P + \beta I \right)^{-1}, \tag{8}$$

with K_e the anchor keys of the current edit, K_p the keys installed earlier in the loop, and R the residuals of value targets optimized at the recorded decision state. PACE changes *which decision state is written*, *when another write is requested*, and *how stalled writes are strengthened*, while the backend's key and solve stay fixed.

4.5 How one repair unfolds

For a bank task in which `apply_credit_card` requires an existing username, a credit score above 600, and prior authentication. After confirming the first two conditions, a ReAct agent produces

```
Observation: credit score = 650
Thought: The score exceeds 600. I can proceed.
Action: apply_credit_card {...}  ×
SOP-valid action: login_user {...}.
```

The checker identifies the omitted authentication step and constructs $e = (c, s, a^-, a^*, \hat{\tau})$: c ends at the credit-score observation, s is the username anchor, a^- is `apply_credit_card`, a^* is `login_user`, and $\hat{\tau}$ is the recorded thought. Optimizing only $p_\theta(a^* \mid c)$ can pass the rewrite probe yet fail when the live agent regenerates the same thought. Decision-state conditioning instead optimizes $p_\theta(a^* \mid c \oplus \hat{\tau})$, aligning the edited value with the state that actually emits the action.

The corrected login is only an intermediate decision. PACE therefore reruns the complete task: if the agent later violates a different requirement, that divergence supplies the next edit and the login key is added to K_p; if it returns to the byte-identical pre-login failure, the controller escalates the existing correction. Thus one incident illustrates both departures from factual editing: the action must be repaired at its deployed decision state, and success must be verified over the remaining trajectory rather than at the edited token alone.

5 Experiments

5.1 Setup

Models, scaffolds, and incidents. We edit Qwen3-4B and Qwen3-8B under ReAct and native function calling. Each setting repairs a domain-balanced set of 30 SOPBench (Li et al., 2025) failures (29 for Qwen3-8B ReAct), with three replicates and at most $B = 5$ installations. The 119 incidents span 28 goals, six domains, and five failure categories. Each edited model is evaluated on 25 held-out tasks. In total, the Qwen experiments contain 2,499 repair loops, 7,500 installed edits, and 62,370 probe rollouts; Appendix G adds GPT-OSS-20B.

Benchmarks. Two similarly named benchmarks play distinct roles. *SOPBench* (Li et al., 2025) supplies every edit set and the own-domain held-out tasks: customer-service domains (`bank`, `dmv`, `healthcare`, `hotel`, `library`, `online_market`) whose SOPs are enforced by an executable checker of action constraints and action graphs. *SOP-Bench-amz* denotes another set of the industrial-SOP tasks (Nandi et al., 2025), whose tools and policy prose are disjoint from SOP-Bench's. It serves as a cross-benchmark held-out probe; all edit sets come from SOPBench.

Metric. A greedy rollout succeeds only if it passes all five SOPBench checks: valid tool calls and constraints, required-action coverage, action-graph order, and the reference final database state. Since every edited item initially fails, edit-set success equals repair rate. Success at budget k counts items repaired within k installations. On held-out tasks, *transfer* measures success on tasks the unedited model failed, and *locality* measures retention on tasks it passed.

Baselines and statistics. We compare with ROME, MEMIT, AlphaEdit, LoRA, FT-M, and GRPO under matched incidents, budgets, execution, and scoring; all methods receive the same checker-mined correction and run in the same feedback loop. PACE uses conditioning plus escalation on Qwen3-4B and escalation alone on Qwen3-8B. We compare PACE with AlphaEdit using a two-sided sign test on per-item success averaged over replicates. Appendix D gives hyperparameters, sampling, probes, and compute.

Paired design and probe coverage. Domains contribute equally to each edit set, and every method and replicate processes the same incidents. The 119 Qwen incidents include all five SOP-Bench failure categories: 62 action-graph violations, 30 premature exits, 13 tool-call errors, 8 constraint violations, and 6 stalls. Replicates rerun the same items, so differences arise from GPU nondeterminism amplified by long-horizon execution; the sign test therefore treats the incident,

Table 1: Trajectory-level SOP success at budget $B = 5$ (mean $\pm$ sd over replicates). All items fail before editing, so success equals the repair rate. PACE runs conditioning and escalation on Qwen3-4B and escalation only on Qwen3-8B. Bold marks the best mean per column. The final two rows report the prespecified paired comparison against the projected-editing baseline as better/worse item counts and two-sided sign-test p-values.

	ReAct		Function calling	
Method	Qwen3-4B	Qwen3-8B	Qwen3-4B	Qwen3-8B
ROME	0.18 ± 0.04	0.38 ± 0.03	0.28 ± 0.10	0.32 ± 0.07
MEMIT	0.19 ± 0.07	0.22 ± 0.05	0.19 ± 0.04	0.35 ± 0.07
LoRA	0.31 ± 0.07	0.45 ± 0.06	0.17 ± 0.07	0.23 ± 0.09
FT-M	0.23 ± 0.00	0.07 ± 0.00	0.13 ± 0.00	0.03 ± 0.00
GRPO	0.31 ± 0.07	0.43 ± 0.04	0.12 ± 0.02	0.43 ± 0.03
AlphaEdit	0.28 ± 0.08	$\mathbf{0.51 \pm 0.07}$	0.39 ± 0.02	0.44 ± 0.04
PACE	$\mathbf{0.50 \pm 0.09}$	$\mathbf{0.51 \pm 0.05}$	$\mathbf{0.52 \pm 0.11}$	$\mathbf{0.66 \pm 0.04}$
PACE vs. AlphaEdit	14 / 1	6 / 6	9 / 1	13 / 2
Sign test p	0.001	1.0	0.02	0.007

rather than the rollout, as the paired unit. For each edited item, the final model is evaluated on 10 own-domain SOPBench tasks, five τ^2-bench telecom tasks, and ten SOP-Bench-amz tasks. The own-domain and SOP-Bench-amz probes include both previously failing tasks for transfer and previously passing tasks for locality, while telecom is transfer-only. Probe draws are seeded by incident identifier, ensuring that every method is evaluated on identical item–task pairs.

5.2 Main results

End-to-end SOP success (Table 1). PACE has the highest or tied-highest mean success in every setting: 0.50 and 0.51 under ReAct, and 0.52 and 0.66 under function calling, for Qwen3-4B and Qwen3-8B respectively. The prespecified paired test is significant in three of four settings ($p \leq 0.02$); the methods tie at 0.51 on Qwen3-8B ReAct. In that setting, escalation fires less often because it requires a byte-identical re-divergence, whereas regenerated `Thought` text often changes the deploy context. PACE's performance is stable across scaffolds, while several training baselines degrade under function calling: LoRA falls from 0.31 and 0.45 under ReAct to 0.17 and 0.23, and GRPO on Qwen3-4B falls from 0.31 to 0.12 (Appendix F).

Mechanism-level pattern. On Qwen3-8B, where PACE uses escalation without conditioning, it reaches 0.66 under function calling and 0.51 under ReAct. The larger gain under function calling is consistent with the trigger design: repeated divergences are more common when the model emits a tool call directly than when it regenerates a free-form `Thought`. On Qwen3-4B, the configuration with conditioning and escalation reaches 0.50 under ReAct and 0.52 under function calling.

Closed-loop gains (Figure 2). As the budget grows from one to five edits, PACE rises from 0.27 to 0.50 on Qwen3-4B and from 0.31 to 0.51 on Qwen3-8B under ReAct. Under function calling, it rises from 0.30 to 0.52 and from 0.31 to 0.66 on Qwen3-4B and Qwen3-8B, respectively. Thus, most of the attainable success lies beyond the first corrected decision, confirming that procedural repair requires following the trajectory after each intervention rather than stopping at a locally corrected action. Normalized area under the budget curve likewise places PACE first in three settings and tied for first in the fourth (Appendix F).

5.3 Locality, transfer, and general capability

Locality and transfer (Figure 3). PACE retains 0.81 of previously passing own-domain tasks under ReAct and 0.86 under function calling; it retains 0.85–0.86 on SOP-Bench-amz in both scaffolds. By contrast, LoRA and GRPO retain only 0.43/0.28 and 0.31/0.29 of previously passing own-domain tasks under ReAct/function calling. The methods with the highest transfer to SOP-Bench-amz, LoRA under ReAct (0.43) and GRPO under function calling (0.38), therefore incur

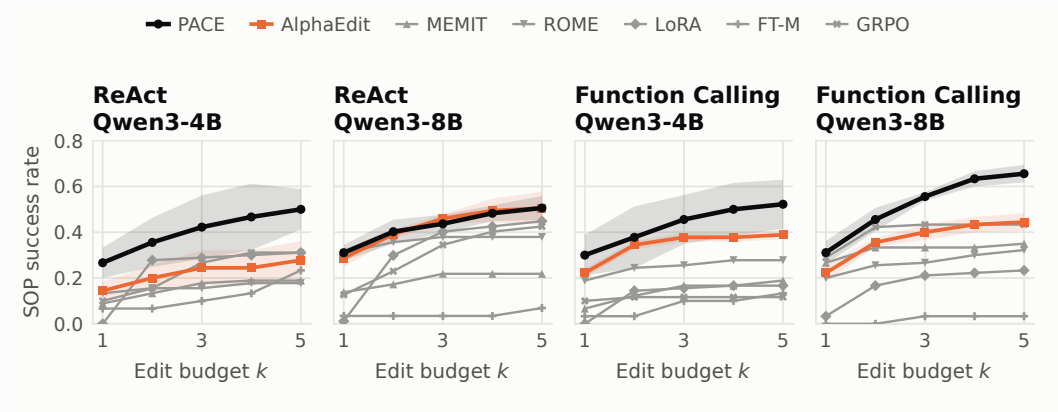

Figure 2: Trajectory-level SOP success across edit budgets $k = 1, \ldots, 5$. The four panels show ReAct on Qwen3-4B and Qwen3-8B, followed by native function calling on the same models. Lines show replicate means; shaded bands indicate ± 1 standard deviation. PACE is highlighted in black.

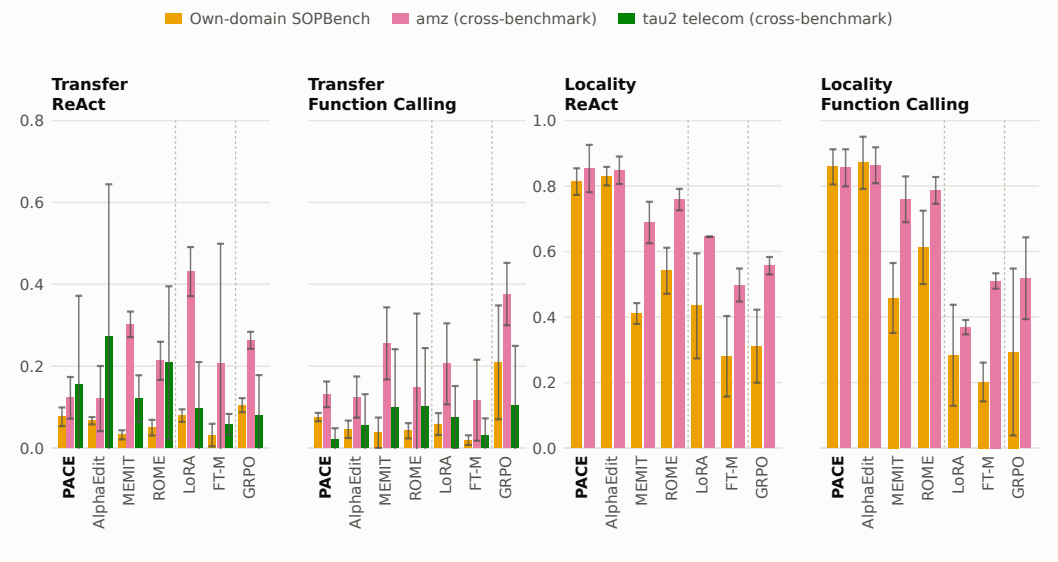

Figure 3: Held-out transfer and locality, pooled over Qwen3-4B and Qwen3-8B. From left to right: transfer under ReAct, transfer under native function calling, locality under ReAct, and locality under native function calling. Transfer measures success on previously failing tasks; locality measures retention on previously passing tasks. Colors distinguish own-domain SOPBench, SOP-Bench-amz, and τ^2-bench telecom; in the figure, "amz" refers to SOP-Bench-amz. Telecom has no locality split. Bars show means and whiskers show ± 1 standard deviation. The SOP-Bench-amz probe is rolled out under ReAct in both scaffolds, so its function-calling bars also cross scaffolds.

substantially greater interference. PACE changes the edited procedure while largely preserving what already worked.

General capability (Table 2). On a five-task probe (GSM8K, BoolQ, Natural Questions, natural language inference, and MMLU) (Cobbe et al., 2021; Clark et al., 2019; Kwiatkowski et al., 2019; Dagan et al., 2005; Hendrycks et al., 2021), PACE's post-edit change ranges from -0.09 to $+0.01$ points, while LoRA and GRPO lose up to 16.39 and 3.55 points.

Table 2: General-capability preservation. Cells report post-edit minus pre-edit macro accuracy (percentage points, mean ± SE over edited items, two ReAct replicates); values near zero indicate preservation.

	Qwen3-4B		Qwen3-8B	
Method	$B = 1$	$B = 5$	$B = 1$	$B = 5$
PACE	-0.09 ± 0.05	-0.09 ± 0.05	$+0.01 \pm 0.01$	-0.07 ± 0.02
AlphaEdit	-0.10 ± 0.04	-0.10 ± 0.05	-0.03 ± 0.01	-0.03 ± 0.03
ROME	-0.07 ± 0.08	-1.54 ± 0.98	$+0.02 \pm 0.04$	-0.05 ± 0.12
MEMIT	-0.06 ± 0.05	$+0.01 \pm 0.08$	-0.20 ± 0.08	-0.49 ± 0.11
LoRA	-16.39 ± 1.77	-1.74 ± 0.35	-8.56 ± 1.26	-0.16 ± 0.14
GRPO	-3.55 ± 0.42	-3.28 ± 0.65	-2.87 ± 0.45	-2.82 ± 0.65

5.4 Data and compute efficiency

PACE uses one checker-identified divergence per correction and no auxiliary training corpus. Under ReAct, it installs an edit in 32 and 86 seconds on Qwen3-4B and Qwen3-8B; under function calling it takes 27–30 seconds, about 3× faster than GRPO. Appendix B reports the full timing comparison.

6 Conclusion

This work reframes model editing for agents as a closed-loop, trajectory-level problem. A correction is not successful merely because the model assigns high likelihood to a replacement under teacher forcing: it must change the action sampled from the agent's actual decision state and allow the subsequent trajectory to remain procedure-compliant. This distinction exposes two failure modes of conventional factual editing: installed-but-inert corrections caused by a scoring-site/sampling-site mismatch, and downstream violations that remain after the first decision is repaired. PACE addresses both by conditioning edits on the recorded decision state, rerunning the agent after every update, and using the resulting trajectory either to select the next divergence or to strengthen an unchanged correction. PACE attains the highest or tied-highest mean SOP success in all four settings, reaching 0.50–0.51 under ReAct and 0.52–0.66 under function calling. The improvement from one to five edits shows that execution feedback is not only a verification step but also a source of sequential supervision. At the same time, PACE retains 0.81–0.86 of previously passing own-domain tasks, and installs edits about 3× faster than GRPO under function calling. These results establish procedural editing as a distinct and useful complement to runtime enforcement: given a reliable mechanism for localizing violations and specifying valid replacements, failures encountered during execution can be converted into targeted, persistent, and behaviorally verified policy repairs. Extending this approach to noisier feedback, stochastic rollouts, and continually accumulated edits is an important next step toward deployment-scale procedural adaptation.

7 Limitations

PACE requires a step-level signal that localizes the first divergent decision and supplies a valid replacement action; a task-level outcome alone provides neither. SOPBench supplies this signal through an executable checker, reflecting regulated workflows with formalized, audited procedures. However, the controller consumes only the local tuple $d = (c, a^-, a^*)$, not the full procedure, so the same loop could use LLM judges over policy documents, process reward models, reference trajectories, or human annotations. Evaluating these alternatives is necessary to extend PACE to procedures specified only in prose. The current implementation uses one null-space-projected editing backend. We evaluate Qwen3-4B and Qwen3-8B under ReAct and function calling, with GPT-OSS-20B as an additional ReAct setting. These experiments establish the method across model scales and scaffolds but leave other editing backends, larger models, and additional agent architectures for future evaluation. Finally, the study uses deterministic greedy rollouts and a five-edit budget to isolate procedural repair. Evaluating stochastic policies, longer repair horizons, and continual edits accumulated across many deployed tasks would further characterize robustness at operational scale.

ETHICS STATEMENT

This work edits the weights of LLM agents that operate under safety- and compliance-relevant procedures. All experiments use public benchmarks with simulated users and synthetic records; no human subjects or personal data are involved. Because an edit installs the preference specified by the SOP checker, deployments should audit the checker and log cumulative edits so that they can be rolled back.

REPRODUCIBILITY STATEMENT

Code, edit sets, and hyperparameter files are available at the anonymous repository linked in the abstract. Appendix C gives the backend derivations, Section 4.3 and Algorithm 1 specify the controller and escalation rung, and Appendix D specifies item sampling, held-out probes, replicates, and compute.

AI USE STATEMENT

Generative AI tools were used only to proofread the manuscript and improve its grammar and readability. They were not used to generate research ideas, methods, experiments, code, data, analyses, interpretations, or scientific claims. The authors reviewed all AI-assisted edits and take full responsibility for the final content of the paper.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Guiding LLMs the right way: Fast, non-invasive constrained generation. In *International Conference on Machine Learning*, 2024.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *ArXiv preprint*, abs/2110.14168, 2021. URL https://arxiv.org/abs/2110.14168.

Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In *Machine learning challenges workshop*, pp. 177–190. Springer, 2005.

Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Xiang Wang, Xiangnan He, and Tat-Seng Chua. AlphaEdit: Null-space constrained knowledge editing for language models. In *International Conference on Learning Representations*, 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Baixiang Huang et al. Model editing as a double-edged sword: Steering agent ethical behavior toward beneficence or harm. In *AAAI Conference on Artificial Intelligence*, 2026.

Terry Koo, Frederick Liu, and Luheng He. Automata-based constraints for language model decoding. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026.

Zekun Li et al. SOPBench: Evaluating language agents at following standard operating procedures and constraints. In *International Conference on Learning Representations*, 2025.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems*, volume 35, pp. 17359–17372, 2022.

Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. In *International Conference on Learning Representations*, 2023.

Subhrangshu Nandi et al. SOP-Bench: Complex industrial SOPs for evaluating LLM agents. *arXiv preprint arXiv:2506.08119*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Song Wang, Yaochen Zhu, Haochen Liu, Zaiyi Zheng, Chen Chen, and Jundong Li. Knowledge editing for large language models: A survey. *ACM Computing Surveys*, 2024a. arXiv:2310.16218.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Anbang Ye et al. SOP-Agent: Empower general purpose AI agent with domain-specific SOPs. *arXiv preprint*, 2025.

Ningyu Zhang, Yunzhi Yao, Bozhong Tian, Peng Wang, Shumin Deng, Mengru Wang, Zekun Xi, Shengyu Mao, Jintian Zhang, Yuansheng Ni, Siyuan Cheng, Ziwen Xu, Xin Xu, Jia-Chen Gu, Yong Jiang, Pengjun Xie, Fei Huang, Lei Liang, Zhiqiang Zhang, Xiaowei Zhu, Jun Zhou, and Huajun Chen. A comprehensive study of knowledge editing for large language models. *arXiv preprint arXiv:2401.01286*, 2024.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

A Worked Example: Skipped Authentication

In a bank-domain credit-card task, `apply_credit_card` requires an existing username, a credit score above 600, and prior authentication through `login_user`. The following Qwen3-8B ReAct trajectory illustrates the failure. Although this example uses Qwen3-8B, the experiments enable conditioning for Qwen3-4B. The agent verifies the first two conditions and then commits the violation:

```
Action: internal_check_username_exist {"username":
"jane_doe"}
Observation: True
Action: internal_get_credit_score {"username": "jane_doe"}
Observation: 650
Thought: The score exceeds 600. I can proceed.
Action: apply_credit_card {"username": "jane_doe", ...}  ×
SOP-valid action: login_user {"username": "jane_doe", ...}
```

The agent has the correct tools and valid arguments but drops one conjunctive precondition. The miner emits $e = (c, s, a^-, a^*, \hat{\tau})$, where c ends at the credit-score observation, s is the username span `jane_doe`, whose last occurrence in c provides the key, a^- is `apply_credit_card`, a^* is `login_user`, and $\hat{\tau}$ is the recorded rationale. A value optimized at $p_\theta(a^* \mid c)$ can pass its teacher-forced probe while the agent regenerates the same rationale and action; conditioning instead optimizes $p_\theta(a^* \mid c \oplus \hat{\tau})$. After login, the agent must continue the workflow and may diverge again, so PACE reruns the task and converts a later divergence into another protected edit. A byte-identical return to the pre-login state instead triggers escalation.

B Edit-Installation Cost

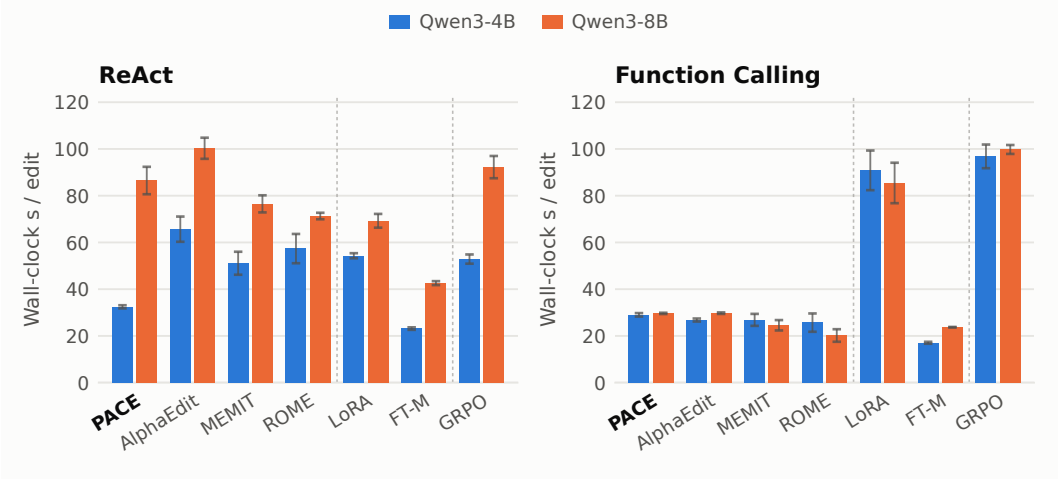

Figure 4: Wall-clock edit-installation cost in seconds per installed edit under ReAct (left) and function calling (right). Bars show replicate means and ± 1 standard deviation.

C Editing-Backend Derivations

Following ROME (Meng et al., 2022) and MEMIT (Meng et al., 2023), we treat the feed-forward down-projection $W^{(\ell)} \in \mathbb{R}^{d \times m}$ of transformer layer ℓ as a linear associative memory: for a key $k \in \mathbb{R}^m$, it emits the value $v = W^{(\ell)} k$. Preserved keys are summarized by the uncentered second moment $C_0 = \mathbb{E}_{k_0 \sim \mathcal{D}_0}[k_0 k_0^\top]$.

ROME enforces a new association while preserving old ones in the C_0-weighted least-squares sense:

$$\min_{\Delta} \quad \left\| \Delta C_0^{1/2} \right\|_F^2 \quad \text{s.t.} \quad (W + \Delta)k^* = v^*, \tag{9}$$

$$\Rightarrow \quad \Delta_{\text{ROME}} = \frac{(v^* - Wk^*)\,(C_0^{-1}k^*)^\top}{(k^*)^\top C_0^{-1} k^*}. \tag{10}$$

The rank-one update is exact on k^*, but preservation is soft and sequential edits carry no mutual protection. MEMIT generalizes to batches and a band of layers:

$$\Delta = R K_e^\top \left(\lambda C_0 + K_e K_e^\top\right)^{-1}, \tag{11}$$

where K_e stacks the edit keys and R stacks their residuals; the update is distributed over the remaining layers.

AlphaEdit converts preservation from a penalty into an invariant. Diagonalizing $C_0 = U\Sigma U^\top$, let $P = \hat{U}\hat{U}^\top$ project onto the approximate null space formed by eigenvectors with eigenvalues below a threshold ϵ. With previously installed edit keys K_p additionally protected, the objective

$$\min_{\Delta} \; \|(W + \Delta P)K_e - V_e\|_F^2 + \beta \|\Delta P\|_F^2 + \|\Delta P K_p\|_F^2 \tag{12}$$

admits

$$\Delta_{\text{AlphaEdit}} = R K_e^\top P \left(K_e K_e^\top P + K_p K_p^\top P + \beta I\right)^{-1}. \tag{13}$$

PACE inherits this solve unchanged; it changes the state at which R is optimized and uses execution feedback to determine subsequent writes.

D Experimental Data, Probes, and Compute

Data and paired design. Edit requests are mined from recorded failures using 30-item subsets drawn five per domain across `bank`, `dmv`, `healthcare`, `hotel`, `library`, and `online_market`. We use one subset per (model, scaffold) because a failure trajectory belongs to the model that produced it. Qwen3-8B ReAct yields four minable library failures, so its subset has 29 items. The edit sets exclude `university`, which has six should-succeed tasks and yields at most one minable failure for most models. We balance domains because base compliance is strongly domain-dependent, from 27.7% to 64.2% across domains for Qwen3-4B (Table 4); balancing prevents a method that repairs one procedure family from dominating the aggregate. The unit of measurement is a closed repair loop scored by live agentic rollouts: each item runs up to five sequential edits, each followed by a full re-rollout and re-mining step, and then 25 held-out probe rollouts. One (setting, method) cell therefore contains 90 repair loops over three replicates and about 2,250 probe rollouts. The four Qwen edit sets cover 119 distinct incidents over 28 user goals and all five SOPBench failure categories (62 action-graph violations, 30 premature exits, 13 tool-call errors, 8 constraint violations, and 6 stalls). Replicates rerun the same items, so significance is assessed per item with a sign test on replicate-averaged success; the between-replicate standard deviation is 0.02–0.11.

Held-out probes. Probes score tasks other than the edited task on the final cumulatively edited model. Per edited item, we draw 10 own-domain SOPBench tasks from the should-succeed pool (half sharing the edited item's user goal, half not), excluding tasks in the edit set. Each task counts toward transfer or locality according to whether the unedited model failed or passed it. We also draw five τ^2-bench telecom tasks (Barres et al., 2025) (transfer only: every telecom tuple is mined from a failed rollout, so no locality axis exists) and five transfer plus five locality tasks from SOP-Bench-amz (Section 5.1). The SOP-Bench-amz probe is rolled out under ReAct in both scaffolds, so for function-calling edits it also crosses scaffolds. Draws are seeded by the edited item identifier, so every method scores identical (item, task) pairs. For each item, we draw a sample from the pool (e.g., 5 of 12–18 telecom tasks and 5 of 21–22 SOP-Bench-amz transfer tuples for the Qwen models).

Scale and compute. Table 3 reports the size and cost of the main experiments. Each (model, scaffold, method, replicate) run uses one H100 GPU, and GPU-hours are summed from the wall-clock runtime recorded in each run. The totals cover the seven evaluated methods (PACE, AlphaEdit, ROME, MEMIT, LoRA, FT-M, and GRPO). The four Qwen settings total 1,014 GPU-hours, and adding GPT-OSS-20B under ReAct brings the total to 2,181 GPU-hours. Held-out probing accounts for 70% of this time, edit installation for 22%, and rollouts of the edited item for 7%. A single PACE replicate over one edit set takes 7–10 GPU-hours on Qwen and about 63 on GPT-OSS-20B.

Table 3: Scale of the main experiments, summed over three replicates and the seven evaluated methods. *Loops* are item-level repair loops; *Edits* are installed edits, each followed by a live rollout; *Probes* are held-out rollouts on own-domain SOPBench, τ^2-bench telecom, and SOP-Bench-amz tasks. The last column is the mean cost of one PACE replicate.

					Probe rollouts				PACE
Scaffold	Model	Items	Loops	Edits	SOPBench	telecom	SOP-Bench-amz	GPU-h	GPU-h/rep
ReAct	Qwen3-4B	30	630	2,138	6,195	3,150	6,300	236	9.7
ReAct	Qwen3-8B	29	609	1,921	6,090	3,045	6,090	242	10.4
FC	Qwen3-4B	30	630	1,863	6,300	3,150	6,300	264	7.1
FC	Qwen3-8B	30	630	1,578	6,300	3,150	6,300	272	8.0
Qwen total		119	2,499	7,500	24,885	12,495	24,990	1,014	–
ReAct	GPT-OSS-20B	30	630	2,184	6,290	2,250	6,300	1,167	63.2
All		149	3,129	9,684	31,175	14,745	31,290	2,181	–

Setting. The SOP checker supplies PACE's target action by identifying the first invalid transition and a valid replacement. The experiments use benchmark-provided procedures and deterministic greedy rollouts. Null-space projection limits interference, and the held-out probes measure preservation empirically.

E Overall and Per-Domain SOP Success Rates

Table 4: Overall and per-domain SOP success rates of the unedited models under the ReAct scaffold. Overall rates include the number of successful tasks out of 830; domain columns report success percentages. †`university` is excluded from the edit sets (Appendix D).

Model	SOP success rate	bank	dmv	healthcare	hotel	library	online_market	university†
Qwen3-4B	51.08% (424/830)	64.2%	61.9%	62.1%	27.7%	63.6%	52.3%	35.7%
Qwen3-8B	63.25% (525/830)	56.0%	78.4%	73.4%	43.1%	63.6%	74.4%	69.0%
GPT-OSS-20B	70.00% (581/830)	73.1%	70.1%	75.0%	64.1%	63.6%	67.4%	92.9%

F Supplementary Scaffold and Budget Analyses

Cross-scaffold comparison. PACE is the top method under both scaffolds, reaching 0.50 under ReAct and 0.52 under function calling on Qwen3-4B, and 0.51 and 0.66 on Qwen3-8B. Baseline rankings shift with the scaffold: LoRA falls from 0.31 and 0.45 under ReAct to 0.17 and 0.23 under function calling, and GRPO on Qwen3-4B falls from 0.31 to 0.12.

Area under the edit-budget curve. Normalized area under the success-versus-budget curve summarizes $k = 1, \ldots, 5$. PACE attains 0.40 and 0.43 on Qwen3-4B and Qwen3-8B under ReAct, and 0.43 and 0.52 under function calling. It leads across the budget in three settings and ties for the lead on Qwen3-8B ReAct.

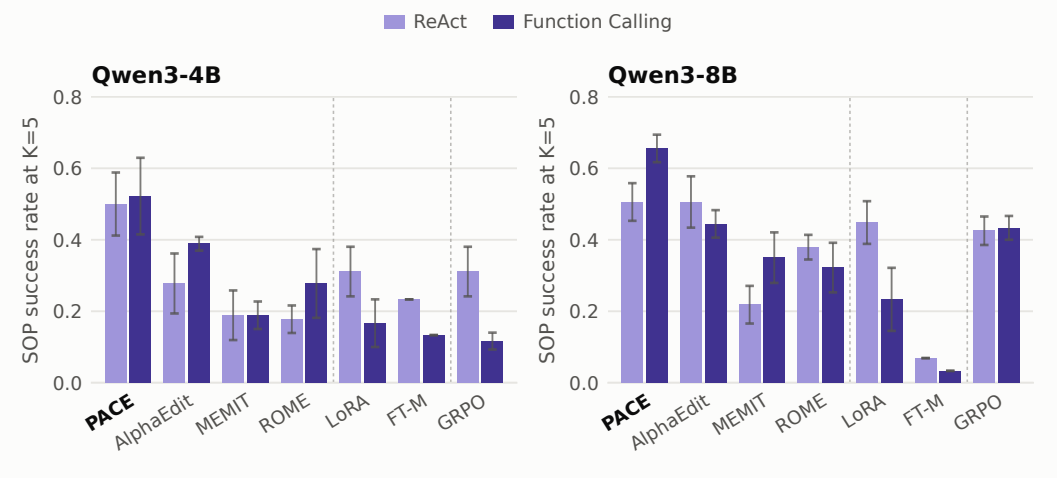

Figure 5: Trajectory-level SOP success at $B = 5$ under ReAct and native function calling. Bars show replicate means and ± 1 standard deviation.

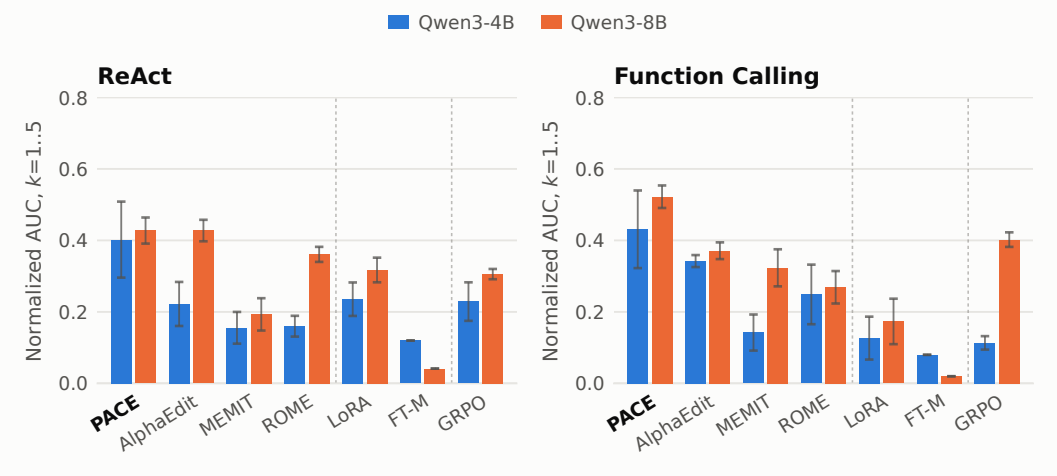

Figure 6: Normalized area under trajectory-level SOP success over edit budgets $k = 1, \ldots, 5$. Bars show replicate means and ± 1 standard deviation.

G SOP Success Across Edit Budgets and Models

On GPT-OSS-20B under ReAct, PACE leads every evaluated method at every budget and reaches 48.9% at $B = 5$ (Table 5).

Table 5: Trajectory-level SOP success rates (%) under ReAct at edit budgets $k = 1, \ldots, 5$ (mean $\pm$ sd over replicates). The Qwen results supplement Table 1 and Figure 2; GPT-OSS-20B provides an additional model. Bold marks the best mean per model and budget.

Model	Method	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
Qwen3-4B	GRPO	10.0 ± 3.3	15.6 ± 3.8	26.7 ± 8.8	31.1 ± 6.9	31.1 ± 6.9
	FT-M	6.7 ± 0.0	6.7 ± 0.0	10.0 ± 0.0	13.3 ± 0.0	23.3 ± 0.0
	ROME	13.3 ± 3.3	15.6 ± 1.9	15.6 ± 1.9	17.8 ± 3.8	17.8 ± 3.8
	MEMIT	8.9 ± 1.9	13.3 ± 3.3	17.8 ± 5.1	18.9 ± 6.9	18.9 ± 6.9
	LoRA	0.0 ± 0.0	27.8 ± 5.1	28.9 ± 5.1	30.0 ± 6.7	31.1 ± 6.9
	AlphaEdit	14.4 ± 3.8	20.0 ± 5.8	24.4 ± 7.7	24.4 ± 7.7	27.8 ± 8.4
	PACE	$\mathbf{26.7 \pm 6.7}$	$\mathbf{35.6 \pm 10.7}$	$\mathbf{42.2 \pm 13.9}$	$\mathbf{46.7 \pm 14.5}$	$\mathbf{50.0 \pm 8.8}$
Qwen3-8B	GRPO	12.6 ± 2.0	23.0 ± 2.0	34.5 ± 0.0	40.2 ± 2.0	42.5 ± 4.0
	FT-M	3.4 ± 0.0	3.4 ± 0.0	3.4 ± 0.0	3.4 ± 0.0	6.9 ± 0.0
	ROME	$\mathbf{31.0 \pm 3.4}$	35.6 ± 2.0	37.9 ± 3.4	37.9 ± 3.4	37.9 ± 3.4
	MEMIT	13.8 ± 3.4	17.2 ± 3.4	21.8 ± 5.3	21.8 ± 5.3	21.8 ± 5.3
	LoRA	1.1 ± 2.0	29.9 ± 8.0	40.2 ± 2.0	42.5 ± 2.0	44.8 ± 6.0
	AlphaEdit	28.7 ± 4.0	39.1 ± 2.0	$\mathbf{46.0 \pm 2.0}$	$\mathbf{49.4 \pm 5.3}$	$\mathbf{50.6 \pm 7.2}$
	PACE	$\mathbf{31.0 \pm 3.4}$	$\mathbf{40.2 \pm 5.3}$	43.7 ± 4.0	48.3 ± 3.4	$\mathbf{50.6 \pm 5.3}$
GPT-OSS-20B	GRPO	7.8 ± 5.1	10.0 ± 6.7	10.0 ± 6.7	11.1 ± 5.1	11.1 ± 5.1
	FT-M	17.8 ± 5.1	23.3 ± 0.0	26.7 ± 0.0	28.9 ± 3.8	33.3 ± 0.0
	ROME	13.3 ± 3.3	15.6 ± 1.9	15.6 ± 1.9	15.6 ± 1.9	15.6 ± 1.9
	MEMIT	20.0 ± 5.8	31.1 ± 6.9	36.7 ± 5.8	41.1 ± 9.6	43.3 ± 8.8
	LoRA	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1.1 ± 1.9	1.1 ± 1.9
	AlphaEdit	23.3 ± 3.3	32.2 ± 6.9	34.4 ± 5.1	43.3 ± 0.0	46.7 ± 0.0
	PACE	$\mathbf{24.4 \pm 5.1}$	$\mathbf{36.7 \pm 3.3}$	$\mathbf{42.2 \pm 1.9}$	$\mathbf{47.8 \pm 3.8}$	$\mathbf{48.9 \pm 5.1}$