Compiling Tool-Use Skills into LoRA Adapters with Multi-Turn Trajectory Replay

Anonymous ACL submission

Abstract

A verified external skill is a curated human-authored procedure that a tool-use agent loads at install time. Such skills reliably raise the rate at which agents produce verifier-accepted artifacts, yet the skill stays as text on disk and re-enters the agent's context at every step. We ask whether such a skill can instead be *compiled* into a per-task LoRA adapter so that its content travels through parameters rather than context. Our study uses the official SkillsBench 72-task suite with OpenCode and Qwen3.6-27B under a strict task–seed paired protocol. After a benchmark calibration that lifts the verifier-passable pool from 39 to 60 tasks, in-context skills prove directional, contributing $+12$ clean gains and 0 pass-level regressions on the paired benchmark. We then introduce a compile pipeline whose central data ingredient is a multi-turn *trajectory-replay* regime that mirrors the agent's ReAct cadence. The audited tally is **7 pass-level wins and 2 partial gains**, with 2 wins where compile also exceeds the with-skill condition on the matched cell. A wrong-adapter control returns reward 0 in $3/3$ tested wins, ruling out the "any adapter suffices" explanation. A paired audit additionally reclassifies two earlier-claimed wins as baseline-pass cells, showing that gate-crossing protocols on similar benchmarks require paired baselines per task–seed cell. We release the harness, the benchmark calibration, the trajectory-replay recipe, the paired audit pipeline, and all per-trial tool-call traces.

1 Introduction

External procedural skills are human-authored, verifier-checkable instructions for solving a specific class of tasks, and they have emerged as a practical interface for extending tool-use agents (Anthropic, 2025; Schick et al., 2023). Recent benchmarks (Li et al., 2026; Jimenez et al., 2024) and skill-discovery systems (Zheng et al., 2025; Wang et al., 2024) converge on the same observation that handing an agent a curated skill measurably raises the rate at which it produces verifier-accepted artifacts. The artifact, however, stays explicit. The skill lives as text on disk, re-enters the agent's working context at every step, and depends on the agent retrieving the right file at the right moment.

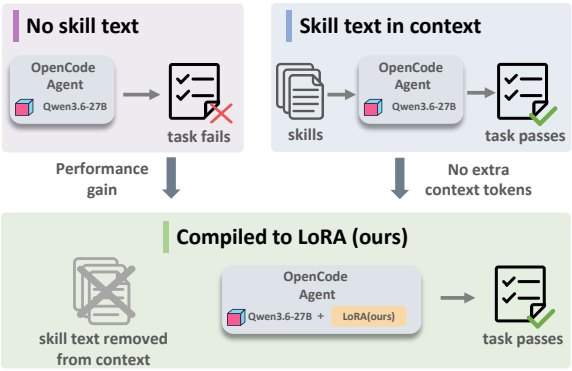

Figure 1: Three settings for delivering a verified skill to a tool-use agent. *Top-left*: without the skill, the agent fails. *Top-right*: with the skill text in context, the agent succeeds at the cost of repeatedly loading the skill text. *Bottom*: we compile the skill into a per-task LoRA adapter, remove the skill text from the agent's context, and ask whether the adapter recovers the with-skill behavior.

A natural question follows. If a skill has already been verified to produce reliable artifacts on a class of tasks, is it worth *compiling* that skill into the model's parameters, for example as a per-task LoRA adapter (Hu et al., 2022; Houlsby et al., 2019; Dettmers et al., 2023), and loading it at inference time instead of feeding the skill text? Compilation in principle removes the per-step context overhead that skill text imposes, eliminates retrieval brittleness, and lets one base model multiplex many skills through inexpensive adapter swaps. The cost is that compilation must actually transfer the *operational* content of the skill, and not just its surface vocabulary, into the adapter's weights.

Figure 1 illustrates the three settings this paper compares: the *no-skill* baseline (skill text removed), the *with-skill* condition (skill text in context), and the *compile* condition (skill compiled into a per-task LoRA, skill text removed). We hold the agent (OpenCode), the backbone (Qwen3.6-27B), and the benchmark (SkillsBench's official 72-task suite) fixed, and ask whether the compile condition can recover the verifier-pass behavior that the with-skill condition provides.

The compile question is hard for three intertwined reasons. Training-record shape matters because supervised fine-tuning records that ignore the agent's rollout shape inject a strong surface bias that does not transfer to the agent's actual read-then-edit cadence. Capacity matters because naively raising LoRA rank does not stabilize the gain; at the rank where compile clears the per-task pass threshold, higher ranks in fact remove the win, suggesting that the bottleneck is not knowledge volume. Measurement matters because the no-skill baseline itself need not be constant across runs, so a baseline that is not paired with the compile condition cell-by-cell cannot decide whether a compile-condition pass is a gate crossing or a coincidence on agent benchmarks with all-or-nothing verifier rewards.

In response, we build an end-to-end compile pipeline (Figure 2) with three stages. The data stage synthesizes training records from a verified skill under five record forms, of which the multi-turn *trajectory-replay* form is our contribution; this form rewrites each verified solution as a sequence of read, tool-result, edit, and tool-result turns that mirror the ReAct cadence (Yao et al., 2023) the agent actually produces. The training stage applies supervised fine-tuning to a per-task LoRA adapter on a frozen base model. The inference stage serves the trained adapter behind the same tool-use agent and pairs every measurement of the three conditions by task and seed, auditing previously single-seed claims against same-seed no-skill and with-skill baselines.

This paper makes four contributions. The first is a paired ablation on the SkillsBench official suite after a 21-task benchmark calibration that lifts the verifier-passable pool from 39 to 60 tasks, on which in-context skills give $+12$ clean gains and 0 pass-level regressions on the paired benchmark. The second is a reusable compile pipeline that comprises five data forms with the multi-turn trajectory-replay regime as our central addition, supervised LoRA training, and a third evaluation condition in the runner. The third is a compile-boundary characterization built from a depth study on a reference task, a data-scale ablation, a 54-task broad-coverage evaluation, and a 3-pair wrong-adapter module-identity control; the audited tally is **7 pass-level wins and 2 partial gains**, with the wrong-adapter control returning reward 0 in $3/3$ tested wins. The fourth is a separable measurement finding that the paired audit reclassifies two previously-reported wins on `mars-clouds-clustering` as baseline-pass cells, showing that gate-crossing protocols on similar benchmarks must collect paired baselines per task–seed cell.

2 Related Work

External verified skills. SkillsBench (Li et al., 2026) frames skills as curated, verifier-checked external artifacts and shows that handing them to a tool-use agent measurably raises pass rates; ContractSkill (Lu et al., 2026) argues that this artifact form is itself the unit of maintenance and recommends explicit preconditions, postconditions, and minimal-patch repair; SkillWeaver (Zheng et al., 2025) demonstrates that agents can discover, organize, and reuse such skills at scale. The practitioner-side Skills framework released by Anthropic (2025) packages this idea as a per-skill folder of `SKILL.md` plus scripts and resources, mirroring exactly the artifact form the present paper attempts to compile. This line establishes both that explicit skills help and that they are the right granularity to discuss. Our paper does not re-litigate whether skills help (Sec 4.1 confirms $+12/0$ on a paired denominator) and instead asks what happens after you have such an artifact.

Non-parametric procedural memory. Reflexion (Shinn et al., 2023) exploits verbal feedback and episodic memory across attempts; ExpeL (Zhao et al., 2024) distills experience into natural-language insights reusable across tasks; Self-Refine (Madaan et al., 2023) iterates on the agent's own output as a separable non-parametric loop; Memp (Fang et al., 2025) emphasizes a maintainable, learnable procedural memory built from past trajectories that explicitly avoids parameter updates. These works collectively argue that much of the reuse value of a skill can live outside the parameters. A reviewer would naturally ask why one would compile at all when explicit memory already covers procedure reuse. We find compilation feasi-

ble but narrow: a fixed rank-16 supervised LoRA recipe applied to 54 reference-solvable tasks at seed 0 clears the pass threshold on 5 of the 47 cells where the no-skill baseline itself fails, while alternative regimes (single-turn diff-replay, cross-task merged adapters) regress or stay flat.

Context-to-parameter compression. A separate line studies pushing external text into lightweight adapters. Doc-to-LoRA (Charakorn et al., 2026) meta-learns a hypernetwork that compiles a long document into a per-document LoRA at inference time; Parametric RAG (Su et al., 2025) extends the same idea to retrieval-augmented generation (Lewis et al., 2020), replacing in-context document injection with on-the-fly LoRA merging in the feed-forward layers. LoraHub (Huang et al., 2024) explores the orthogonal question of composing several task-specific LoRAs through gradient-free coefficient search. Prefix Tuning (Li and Liang, 2021) and the broader PEFT line (Houlsby et al., 2019; Dettmers et al., 2023) pre-date this and constrain the space of edits to a small subspace. The compression target there is *knowledge*; verified external procedural skill is a different target: short, imperative, and judged by an executable verifier on the agent's eventual tool calls.

Code-agent benchmarks and harnesses. Functional-correctness benchmarks such as HumanEval (Chen et al., 2021) measure the model's standalone code completion; repository-level benchmarks such as SWE-bench (Jimenez et al., 2024) and the SWE-agent harness (Yang et al., 2024) extend the protocol to multi-file, multi-turn editing. SkillsBench (Li et al., 2026) sits in this family but isolates the *skill* as the unit of evaluation rather than the model or the harness, which makes it the natural testbed for the compile question.

3 Method

We frame skill compilation as a three-stage pipeline (Figure 2). Given a verified skill and its companion reference solution, we (i) synthesize a small set of supervised training records under five record forms; (ii) train a per-task LoRA adapter via supervised fine-tuning of a frozen base model; and (iii) serve the adapter behind a tool-use agent and evaluate it on the verifier under a paired task–seed protocol. Section 3.1 describes the data forms, Section 3.2 describes the training recipe, and Section 3.3 describes the inference and evaluation setup.

3.1 Training-record synthesis

A verified skill in SkillsBench is a short, imperative procedure paired with a reference solution and a deterministic verifier. The compile question requires turning that artifact into supervised training records whose record form transfers to the agent's actual rollout shape. We define five record forms that vary along the axes of source (skill text vs. reference solution vs. verifier checks) and turn structure (single-turn vs. multi-turn).

The first four forms are single-turn. *Structured skill summaries* treat the curated skill text, lightly templated, as a high-information assistant turn. *Scenario-to-cause pairs* pair each task statement with the expected opening turn. *Test-derived supervision* rewrites each verifier check as a one-step assistant tool call. *Single repair examples* take the file-level diff implied by the reference solution and apply it as a single-turn edit target. Together these four forms capture the static content of the skill, but their record shape never reproduces the read–edit cadence that the agent emits at inference time.

The fifth form, *multi-turn ReAct trajectories*, is the central data contribution of this paper. Each file-level repair from the reference solution is rewritten as a six-turn sequence (system, user, assistant read tool call, user tool result, assistant edit tool call, user tool result) that matches one round of the agent's ReAct loop (Yao et al., 2023). Trajectory replay supplies the LoRA with both *what* to read or edit and *when* to switch between reading and editing, two pieces of operational content that the single-turn forms leave implicit. Per-task corpus sizes range from 7 to 117 records depending on the regime, with 30 as a typical default; data-scale ablations appear in Section 4.3.

3.2 Supervised LoRA fine-tuning

For each task we train a separate LoRA adapter (Hu et al., 2022) on top of a frozen Qwen3.6-27B base model (Qwen Team, 2026; Yang et al., 2025). The objective is the standard supervised fine-tuning loss over assistant-message tokens,

$$\mathcal{L}_{\text{SFT}}(\theta) = -\sum_{t} \log p_\theta(y_t \mid \mathbf{y}_{<t}, \mathbf{x}), \qquad (1)$$

where $\mathbf{x}$ is the system and user context, $\mathbf{y}$ is the assistant target sequence, and θ collects only the LoRA matrices (rank r, scaling $\alpha = 2r$, dropout 0.05) injected into the attention and feed-forward projections of every transformer block.

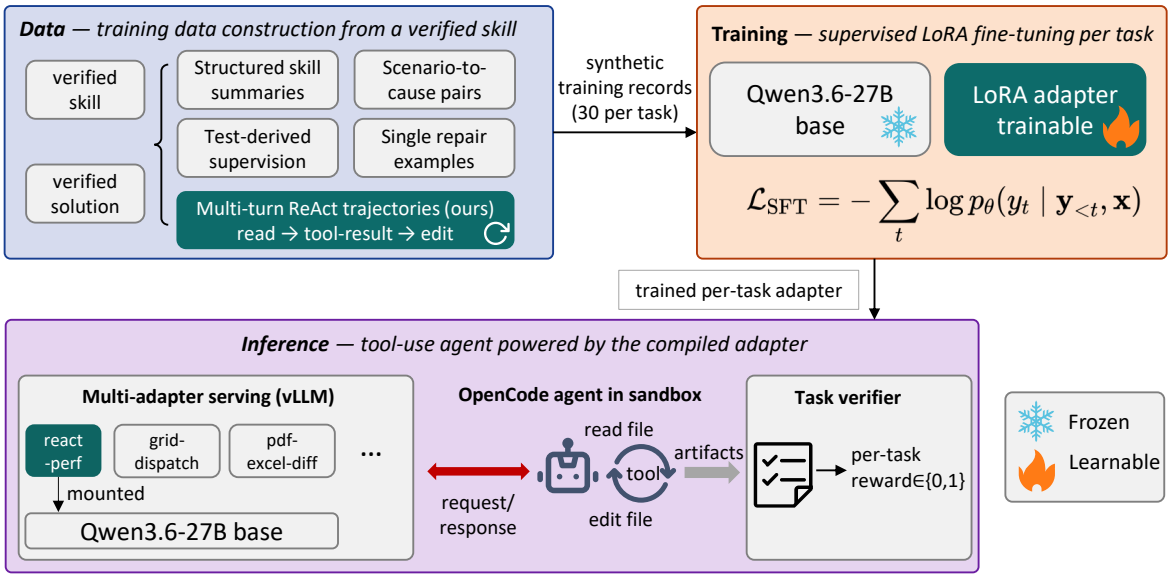

Figure 2: The three-stage compile pipeline. *Data*: from a verified skill and its companion reference solution we synthesize on the order of 30 training records per task across five forms; the multi-turn ReAct trajectory form (highlighted) is the contribution of this paper. *Training*: a per-task LoRA adapter is trained by supervised fine-tuning on top of the frozen Qwen3.6-27B base model. *Inference*: the trained adapter is mounted onto a multi-adapter vLLM server, the agent runs the task in a sandbox, and a per-task verifier returns a $\{0, 1\}$ reward.

We use rank $r{=}16$ by default and explore rank 64 and 128 as ablations (Section 4.2). Optimization uses AdamW with a cosine schedule, peak learning rate 10^{-4}, 5 epochs, batch size 1 with gradient accumulation, and bfloat16 precision. We compare briefly against direct preference optimization (Rafailov et al., 2023) in Section 4.2 and treat on-policy alternatives (Schulman et al., 2017; Ouyang et al., 2022; Bai et al., 2022) as orthogonal future work. Full hyperparameters and per-regime final training losses appear in Appendix A.

3.3 Inference and paired evaluation

At evaluation time the trained adapter is mounted onto a multi-adapter vLLM server (Kwon et al., 2023) alongside the base model and is selected by the agent's task identifier. OpenCode (OpenCode Authors, 2025) runs the task inside the SkillsBench sandbox under the same Harbor orchestrator that handles the no-skill and with-skill conditions, and the task verifier returns a $\{0, 1\}$ reward together with per-test counts.

Every measurement is paired by task and seed. For each task–seed cell the runner produces three trials. The *no-skill* trial removes the skill text from the agent's context, the *with-skill* trial keeps the skill text in context, and the *compile* trial removes the skill text and mounts the per-task LoRA. All three share the same container build, the same verifier, and the same random seed. Pairing eliminates the task-difficulty and infrastructure-flakiness confounders that single-condition leaderboards inherit. We report per-task pass-level reward (1 if the verifier returns full pass, else 0) and the underlying per-test counts. Backbone serving configuration and other implementation details are summarized in Appendix A.

4 Experiments

This section evaluates the compile condition against the no-skill and with-skill baselines under the paired protocol along three axes: depth on a reference task (record-form, rank, cross-task merging, alternative objective), data scale (paraphrase expansion), and coverage (the same recipe applied across all 54 trainable reference-solvable tasks). Section 4.1 first establishes the calibrated benchmark and the in-context skill baseline; Section 4.2 reports the depth study; Section 4.3 reports scale and coverage. All experiments share the same backbone, the same agent, and the same default training recipe (rank-16 LoRA, 5 epochs of supervised fine-tuning) from Section 3; the variation is the form and quantity of training records the LoRA sees.

Paired outcome	#tasks
Clean gain (with-skill passes, no-skill fails)	**12**
Both pass at seed 0	7
Pass-level regression (with-skill $<$ no-skill)	0
Both fail (excluded as upstream-unsolvable)	53
Reference-solvable subtotal	**60**
Total paired	72

Table 1: Paired skill effect on the SkillsBench official suite at seed 0 after benchmark calibration. The reference-solvable subset isolates the 60 tasks where the verifier accepts the reference solution; clean gains are uniformly directional and there are no pass-level regressions.

4.1 Setup

Benchmark calibration. Before any compile experiment is meaningful, we must establish how many SkillsBench tasks the verifier accepts the reference solution on within our offline setup, since unbuildable tasks would otherwise inflate baseline failure rates. In our initial container build, only 39 of 72 tasks were reference-solvable; the remaining 33 failed at infrastructure boundaries (missing package mirrors, host-side binary dependencies absent from the container, vendor-specific solver licences). We triaged 21 of these into six remediation categories and patched them by rewriting package-mirror sources and pre-vendoring the missing binaries into the container, lifting the reference-solvable pool from 39 to 60. The remaining 12 tasks are upstream-unreachable in our setup and are excluded from all compile evaluations.

In-context skill baseline. Across all 72 paired tasks after the remediation, the seed-0 paired ablation yields 12 clean paired gains, 7 both-pass cells, 0 pass-level regressions, and 53 both-fail cells (Table 1). Excluding the 41 both-fail tasks the reference solution itself cannot solve leaves 60 reference-solvable paired tasks with no-skill 7 and with-skill 19, a $+12$ paired delta that is uniformly directional. In-context verified skills are therefore a real, sign-consistent signal on this benchmark, and form the baseline that the compile condition must match or exceed.

4.2 Depth study

Single-turn baseline. The first regime trains a per-task LoRA on `react-performance-debugging` using the four single-turn record forms only (25 records, final training loss 0.137). Under the paired protocol the compile condition fails to reach pass-level reward and at the test level loses 5 tests against no-skill ($45/55$ vs. $50/55$). The failure mode is consistent: the agent reads the relevant source file, declares "the file already uses `Promise.all`, no edit is needed," and never applies the more aggressive batching transformation that the diff-replay records actually encode. Single-turn diff-replay teaches the LoRA *what the fixed file looks like* but not *when to edit*, and the surface bias is strong enough to suppress an otherwise neutral baseline.

Multi-turn trajectory replay. Adding the multi-turn trajectory-replay records (Section 3.1) raises the per-task corpus to 33 records (25 single-turn $+$ 8 trajectory-replay). Under the paired protocol the compile condition reaches pass-level reward on this task (compile $11/11$ tests vs. no-skill $10/11$ and with-skill $10/11$), and test-level totals satisfy compile $50/55 \geq$ no-skill $48/55 \geq$ with-skill $47/55$ over the matched cell. This is the first cell in this study where the compile condition clears the hardest test in the suite that no-skill and with-skill both miss, isolating the contribution of the multi-turn trajectory-replay records over the single-turn baseline.

Rank ablation. Trajectory replay clears the pass threshold at rank 16 with 33 training records. Holding data fixed, we trained two additional LoRAs at rank 64 and rank 128. Training loss drops monotonically with rank, as expected. On the cell where the rank-16 adapter clears the verifier, however, **rank** 64 **regresses to** $9/11$ **and rank** 128 **to** $10/11$ **tests**, and both lose the previously-won hardest test. Rank 128 additionally produces non-Python edits that crash test collection on a separate evaluation cell. Higher rank does not generalize the win; it removes it. The bottleneck is not LoRA capacity.

Cross-task merging. A merged 73-record LoRA combining the reference task with three other close-call tasks (rank 16) produces catastrophic negative transfer on `grid-dispatch-operator`: $0/6$ tests versus the per-task LoRA's $6/6$ at seed 0. The merged LoRA absorbs the reference task's read-then-edit pattern and applies it to a task whose first action should instead be reading a CSV. Single-LoRA multi-skill compilation is not a free upgrade over per-task LoRAs at this scale.

Direct preference optimization. Continuing to train the trajectory-replay LoRA with direct preference optimization (Rafailov et al., 2023) on 12 (chosen, rejected) pairs harvested from the diff-replay trials reaches in-distribution preference accuracy 1.0, but the compile condition then regresses to $0/11$ tests on the matched evaluation cells. Direct preference optimization on 12 in-distribution pairs is not a free lunch at this data scale.

4.3 Scale and coverage

Data-scale ablation. Paraphrased variants of each task's compile records, synthesized through the base model with a strict filter preserving file paths, imports, and numeric thresholds verbatim, expand the reference-task corpus to 117 records. The scaled corpus clears the pass threshold on the matched reference cell (compile $11/11$ vs. no-skill $9/11$ and with-skill $10/11$). On four close-call tasks the same recipe adds zero new pass-level wins, removes one prior win, and on a fifth task regresses the cell from $7/8$ to $0/8$, the largest single-cell test regression in this study.

Broad coverage. A direct response to the "few-task anecdote" concern is to enlarge the experimental footprint. We applied the default per-task supervised LoRA recipe to every reference-solvable task we could prepare automatically and ran a seed-0 smoke against the paired no-skill and with-skill baselines. Four tasks failed automatic data preparation (file-permission errors on the reference solution under our service account); the remaining 54 tasks all trained cleanly, and we evaluated each at seed 0. One task returned an unparseable summary and is recorded as *unparseable*, leaving 53 cells with paired rewards. Table 2 tallies the per-cell outcomes; the two novel-win cells (`pdf-excel-diff` and `pptx-reference-formatting`) exceed both baselines at the same seed.

Wrong-adapter module-identity control. For the three cleanest seed-0 wins from the broad-coverage evaluation we re-ran the compile-only smoke with a deliberately unrelated LoRA mounted in place of the task-matching adapter, keeping every other setting identical (Table 3). All three controls return reward 0. This rules out the "any adapter suffices" alternative explanation: the compile gain depends on the weights of the matching adapter, not merely on the presence of a LoRA. The control is small in scale (3 cells); a stronger version would sweep an $N \times N$ matrix of (target, decoy) pairs, which we leave for follow-up work.

Per-cell outcome (compile vs. baselines)	#cells
Winners that exceed at least one baseline	
Novel win (neither baseline passes)	**2**
Reproducing win (with-skill also passes)	3
Novel partial gain (sub-pass, exceeds both)	2
Baseline-equivalent outcomes	
Baseline-pass cell (no-skill already passes)	7
No-change cell	38
Pass-level regression	1
Excluded from this slice	
Unparseable runner output	1
Reference solution not preparable	18
Total	72

Table 2: Per-cell outcomes for the broad-coverage evaluation at seed 0 (compile condition only). The two novel-win cells exceed both no-skill and with-skill at the same seed.

Target task	Matching adapter	Unrelated adapter
`pdf-excel-diff`	**1**	0
`hvac-control`	**1**	0
`pptx-reference-formatting`	**1**	0
Aggregate	**3/3**	0/3

Table 3: Wrong-adapter module-identity control on three seed-0 wins. The matching adapter clears the verifier; an unrelated adapter mounted in its place returns reward 0 in all three cases.

Audited final tally. Combining depth, breadth, scale, and coverage, the audited grid contains nine task–seed cells where compile beats the paired no-skill at the tested seed (Table 4); these break down into seven pass-level wins and two partial gains. Two of the pass-level wins are *novel*, with compile also exceeding with-skill at the same seed.

5 Analysis

The evidence in Section 4 consistently points to one mechanism. A compiled LoRA reliably absorbs what the agent *says*, including library names, file paths, and tool argument shapes. It absorbs what the agent *does*, that is when to batch reads and when to switch to edits, only when the training-record shape matches the rollout shape the agent will actually emit. Section 5.1 unpacks this mechanism on the reference task; Section 5.2 explains why most broad-coverage cells stay flat despite the same recipe; Section 5.3 draws two measurement

Task	Seed	no-skill	with-skill	compile	Compile recipe
Pass-level wins (7 cells)					
react-perf-debugging	4	0	0	**1**	Trajectory replay r16
hvac-control	0	0	1	**1**	Broad coverage r16
react-perf-debugging	2	0	1	**1**	Scaled replay r16
pdf-excel-diff†	0	0	0	**1**	Broad coverage r16
pptx-reference-formatting†	0	0	0	**1**	Broad coverage r16
mars-clouds-clustering	0	0	1	**1**	Broad coverage r16
software-dependency-audit	0	0	1	**1**	Broad coverage r16
Partial gains (2 cells)					
dialogue-parser	0	0.67	0.67	**0.83**	Broad coverage r16
lab-unit-harmonization	0	0	0	**0.33**	Broad coverage r16

Table 4: Audited tally of compile-condition wins. Columns report per-cell verifier reward at the tested seed. Green-shaded rows (†) are *novel* cells where compile also exceeds with-skill; yellow-shaded rows are sub-pass partial gains. Pass-level columns are binary; partial-gain columns report the fractional pass rate.

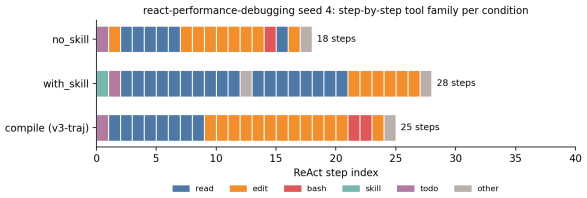

Figure 3: Tool-call traces for the three conditions on react-performance-debugging at seed 4. Each horizontal strip is one rollout; each colored cell is one tool call. The compile trace contains no skill-loading calls and follows a tight read–edit–verify block, mirroring the with-skill plan shape without the skill-loading preamble; the no-skill trace spreads across many edits without converging.

lessons that survive the compile question itself.

5.1 Why trajectory replay works

In the single-turn diff-replay regime the LoRA was trained to map "broken file" → "fixed file" verbatim. At inference time the agent never asks for the whole fixed file; it asks the model to decide *whether* to edit. Surface bias from the diff-replay records pushes the LoRA toward strings like "already uses `Promise.all`, no edit needed," because the assistant target in some training records is structurally similar to that string. The single-turn regime injects what the verifier reads at the end without injecting the decision to make the edit in the first place.

In the trajectory-replay regime, each training record is a read-then-edit-then-verify sequence that matches one round of the agent's ReAct loop (Yao et al., 2023). The cadence injection is sharp enough to clear the verifier on the matched cells of Table 4. Figure 3 shows the effect directly on the reference task: the compile trace adopts the with-skill plan shape but skips the skill-loading preamble entirely. The unifying axis across the four regimes is whether the training-record shape matches the agent's rollout shape.

5.2 Why most cells don't move

To probe why 38 of the 54 broad-coverage cells land at *no change*, we read the per-trial tool-call trace for three representative no-change cells. The compile rollout partially reproduces the with-skill cadence by issuing multiple reads before the first shell command and using a comparable number of steps, yet it *never invokes the canonical skill-loading tool* that with-skill uses to pull the task-specific helper script into the workspace, and writes its own analyzer from scratch. The skill-loading step is the one cadence component that trajectory replay does not transfer, because the training records distill what happens *after* retrieval and the retrieval call itself has no executable hook in the LoRA. The no-change majority therefore reflects a structural absence of skill-helper retrieval, not a failure to absorb cadence.

5.3 Measurement lesson

Mars-clouds reclassification. An earlier version of this paper reported two `mars-clouds-clustering` cells as compile wins, against a no-skill baseline that we had only collected at one seed. Recollecting the paired no-skill and with-skill conditions for the same cells reveals that both baselines themselves reach $8/8$, so the compile-minus-no-skill difference is zero on those cells. After this reclassification the trajectory-replay recipe accounts for the audited tally in Table 4. The lesson is separable from the compile question: a paired baseline at every task–seed cell is needed to detect this kind of baseline behavior before attributing a pass to the adapter.

Module-identity control. The single most informative control in the paper is the wrong-adapter result (Table 3). For the three cleanest seed-0 wins from the broad-coverage evaluation we re-ran the compile-only evaluation with an unrelated adapter mounted in place of the task-matching adapter; all three controls return reward 0. Two readings are compatible. First, module identity is necessary: the win depends on the weights of the matching adapter, not on the mere presence of a LoRA. Second, the unrelated adapters are actively harmful: rather than degrading to the no-skill reward, the controls fall to 0, suggesting that a mismatched

adapter pushes the agent into a rollout shape unrelated to the target task. The control is small in scale (3 cells); a stronger version would sweep an $N \times N$ (target, decoy) matrix and is left for follow-up.

6 Conclusion

This work asked whether a verified external skill can be compiled into a per-task LoRA adapter and still recover the verifier-pass behavior that handing the agent the skill text in context produces. Under a strict task–seed paired protocol on the SkillsBench official suite with OpenCode and Qwen3.6-27B, the multi-turn trajectory-replay regime introduced here yields an audited tally of seven pass-level wins plus two partial gains, with a wrong-adapter control returning reward 0 in $3/3$ tested wins. The same paired audit additionally reclassifies two earlier-reported wins as baseline-pass cells, surfacing a separable measurement finding that gate-crossing protocols on similar benchmarks must collect paired baselines per task–seed cell. We release the harness, the benchmark calibration, the trajectory-replay recipe, the paired audit pipeline, and all per-trial tool-call traces as a measurement baseline for future work.

Limitations

Our study uses a single base model (Qwen3.6-27B) and a single tool-use agent (OpenCode). The trajectory-replay claim should be read against this stack and we do not claim transfer to other backbones or non-ReAct agents.

A practical constraint also surfaces on the serving side. Expanding the LoRA's target-module set beyond attention and feed-forward projections (for example to include the embedding and output projection) made our serving stack reject the trained adapter at load time. The set of modules to which a LoRA may be applied at training time is therefore strictly larger than the set that current multi-adapter serving stacks accept, which couples training and serving choices in a way that is easy to overlook.

Routing is treated as orthogonal in our study. In every trial exactly one adapter is loaded, identified by the trial's task name. The wrong-adapter control of Section 4.3 probes one limited variation by deliberately mismatching the loaded adapter, and all three controls return reward 0. On-the-fly multi-adapter arbitration through skill-card matching or learned routing is the natural follow-up but is empirically separate from the compile question itself.

References

Anthropic. 2025. Introducing agent skills. https://www.anthropic.com/news/skills, accompanying open-source repository at https://github.com/anthropics/skills.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, and 1 others. 2022. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*.

Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. 2026. Doc-to-LoRA: Learning to instantly internalize contexts. *arXiv preprint arXiv:2602.15902*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. *Advances in Neural Information Processing Systems*.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. 2025. Memp: Exploring agent procedural memory. *arXiv preprint arXiv:2508.06433*.

Neil Houlsby, Andrei Giurgiu, Stanisław Jastrzębski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*.

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

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. 2024. LoraHub: Efficient cross-task generalization via dynamic LoRA composition. *Conference on Language Modeling (COLM)*.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Annual Meeting of the Association for Computational Linguistics*.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, and 1 others. 2026. SkillsBench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*.

Zijian Lu, Yiping Zuo, Yupeng Nie, Xin He, Weibei Fan, Lianyong Qi, and Shi Jin. 2026. ContractSkill: Repairable contract-based skills for multimodal web agents. *arXiv preprint arXiv:2603.20340*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*.

OpenCode Authors. 2025. OpenCode: An open-source coding agent. https://github.com/sst/opencode, version 1.14.29 used in this work.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*.

Qwen Team. 2026. Qwen3.6-27B: Flagship-level coding in a 27b dense model. https://qwen.ai/blog?id=qwen3.6-27b.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*.

Weihang Su, Yichen Tang, Qingyao Ai, Junxi Yan, Changyue Wang, Hongning Wang, Ziyi Ye, Yujia Zhou, and Yiqun Liu. 2025. Parametric retrieval augmented generation. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*.

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

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. ExpEl: LLM agents are experiential learners. In *AAAI Conference on Artificial Intelligence*.

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. 2025. SkillWeaver: Web agents can self-improve by discovering and honing skills. In *Conference on Language Modeling (COLM)*.

A Hyperparameters and Final Training Losses

Table 5 reports the training-record counts, LoRA ranks, epochs, and final supervised train-

ing losses for each regime used in this paper. Unless otherwise noted, every regime uses $\alpha = 2r$, dropout 0.05, target modules $\{q, k, v, o, \text{gate}, \text{up}, \text{down}\}_{\text{proj}}$, AdamW with a cosine schedule, learning rate 1×10^{-4}, bfloat16 precision, and maximum sequence length 2048. Per-task adapters in the broad-coverage evaluation use the same default training recipe with per-task record counts in the 7–117 range.

Training regime	Task(s)	Records	Rank	Epochs	Train loss
Depth study on the reference task (Sec. 4.2)					
Single-turn diff-replay	reference	25	16	5	0.137
Trajectory replay (ours)	reference	33	16	5	0.769
Trajectory replay (rank ablation)	reference	33	64	5	0.402
Trajectory replay (rank ablation)	reference	33	128	5	0.322
Cross-task generalization (Sec. 4.2)					
Cross-task single-turn	T1	19	16	5	1.062
Cross-task single-turn	T2	12	16	5	1.273
Cross-task single-turn	T3	9	16	5	1.477
Scale and coverage (Sec. 4.3)					
Scaled trajectory replay	reference	117	16	5	0.142
Broad-coverage evaluation	54 tasks	7–117	16	5	0.14–2.56

Table 5: Training-record counts and final supervised training losses across all regimes used in the study. Rows shaded in blue are the regimes that produce the audited wins of Table 4; "reference" denotes the depth-study task `react-performance-debugging`; T1–T3 are the three close-call companion tasks of the cross-task regime in Section 4.2.