TURNROUTE: LEARNING MODULAR AGENT CAPABILITIES FOR DYNAMIC COMPOSITION

Anonymous authors
Paper under double-blind review

ABSTRACT

Language-model agents coordinate multiple capabilities over long interactions, yet typically learn them jointly within a monolithic set of adapted parameters. Can these capabilities instead be decoupled into separate parameter modules and dynamically recomposed during execution? We introduce TurnRoute, a framework that trains capability-specific LoRA experts through turn-level loss masking of labeled trajectories over a shared frozen backbone. During execution, a router selects one adapter to generate each turn while maintaining a shared interaction history, with an unmasked adapter providing a fallback. On software engineering tasks, capability probes show that masked experts retain comparable competence to monolithic adaptation on their assigned capability while exhibiting behavioral separation across adapters. Although individual specialists underperform as standalone agents, TurnRoute achieves pass@1 of 15.8% at 7B and 28.4% at 14B on 500 SWE-bench Verified instances, outperforming monolithic LoRA by 1.6 and 2.0 pp. It reaches 98% and 92% of full fine-tuning performance with roughly 9.4–$11.7\times$ fewer trainable parameters than full fine-tuning, while activating only one expert adapter per turn for response generation. In addition, at 7B, TurnRoute increases non-empty patch production from 77.8% to 94.4%, reducing empty-patch failures by 74.8%. Together, these results demonstrate the feasibility of decoupling agent capabilities into separately trainable parameter modules and dynamically recomposing them while retaining end-to-end task competence, providing a practical approach to modular adaptation of language-model agents.

1 INTRODUCTION

Language-model agents increasingly act over long interactions, where solving a task requires coordinating multiple capabilities and revisiting them as new observations arrive. Unlike single-turn generation, such agents must maintain a policy over sequences of actions, dynamically deciding what kind of operation is needed next. We study this problem in software engineering, a demanding setting where an agent must navigate unfamiliar repositories, inspect and edit code, execute commands, observe their effects, and revise its approach. SWE-bench formalizes this setting by asking an agent to resolve a GitHub issue in a repository snapshot (Jimenez et al., 2024), while SWE-agent provides the interactive agent-computer interface (Yang et al., 2024).

Such long-horizon interactions call for several complementary capabilities rather than one uniform skill. Following Kang et al. (2026), we define a *capability* as a recurring competence, exercised through one or more trajectory actions, that is necessary to solve a subset of task instances in the target environment. For software engineering (SWE) tasks we fix four: `localization`, `code_generation`, `verification`, and `tool_use`. They do not occur in a fixed order. An agent must locate a faulty function among thousands of files before fixing it, and run the test suite afterwards. A failed test may send it back to localization, and newly inspected code may invalidate an earlier edit (Figure 1). Each capability recurs across tasks and may be revisited multiple times, making SWE a natural testbed for studying capability decomposition and recomposition.

A common way to adapt an agent to such trajectories is to fine-tune a single model (Hu et al., 2022; Pan et al., 2025; Yang et al., 2025a). This treats heterogeneous turns from different capabilities as supervision for the same set of adapted parameters (Figure 1a). As a result, localization, code generation, verification, and other recurring behaviors remain entangled within one monolithic pol-

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.

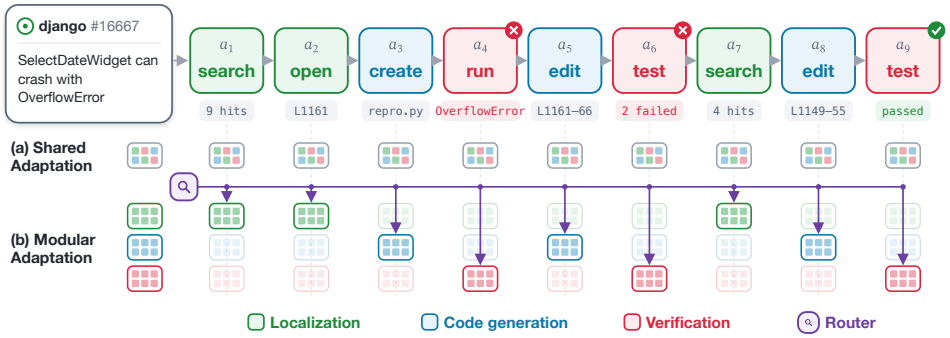


Figure 1: An illustrative agent episode on a SWE-bench Verified issue. Each action exercises one capability. (a) Shared adaptation: the same parameters produce every action, so all capabilities are entangled in one module. (b) Modular adaptation: each capability has its own parameter module, and a router selects one module for each action.

icy: they are not represented as independently trainable modules that can be recomposed during execution. This monolithic parameterization leaves unclear whether the capabilities learned from a trajectory can be represented as modular components of the agent policy.

Existing approaches provide partial forms of capability decomposition. In software engineering, multi-agent systems assign specialized roles through prompts and orchestration (Chen et al., 2024; Arora et al., 2024; Trae Research Team et al., 2025), but give no capability its own trainable parameters. Dynamic adapter-composition methods do train skill-specific parameters, but typically route at the token, layer, or module level, below the granularity of an observable agent action. TRACE comes closest to capability-level modular training, but acquires missing capabilities by synthesizing new environments and training dedicated experts with reinforcement learning (Kang et al., 2026). These approaches leave open whether capabilities already present within agent trajectories can be separated into independently trainable modules and recomposed dynamically in the same trajectory.

This leaves open a central question: **can capabilities learned within agent trajectories be decoupled into separately trainable parameter modules and dynamically recomposed during execution without sacrificing end-to-end competence**? We introduce TurnRoute, a framework for modular adaptation of language-model agents. Given trajectories whose turns can be associated with recurring capabilities, TurnRoute labels turns by capability, trains capability-specialized LoRA experts over a shared frozen backbone, and learns to select one expert for each assistant turn while maintaining a shared interaction history (Figure 1b). We instantiate and evaluate TurnRoute on software-engineering agents, where capability transitions recur within individual trajectories.

In summary, our main contributions are as follows:

- **Modular Capability Adaptation.** We introduce **TurnRoute**, a framework that decomposes capabilities learned from agent trajectories into separately trainable parameter modules and dynamically recomposes them at the turn level during execution.
- **Capability Decoupling.** We show that capability masking preserves competence on the assigned capability while producing behaviorally distinct specialists. Although individual specialists are insufficient as standalone agents, their dynamic recomposition recovers monolithic-level competence and yields slightly higher observed pass@1.
- **Prospective Turn-Level Routing.** We compare Hindsight and Foresight supervision and show that routing based on what the next step should do is more effective than reproducing the capability sequence of the original rollout.

2 Capability Decoupling

Before introducing the full TurnRoute framework, we first ask whether recurring capabilities within an agent trajectory can be separated into capability-specialized trainable parameter modules. We study this question in software engineering, where multiple capabilities repeatedly interleave within the same trajectory. Drawing inspiration from role-decomposed agentic systems (Chen et al., 2024; Arora et al., 2024; Trae Research Team et al., 2025), we formalize SWE trajectories and a four-

Table 1: Capability probes (%), read-only and outside the agent loop. **On-cap.** is the adapter masked for the probed capability, **Mono.** the monolithic LoRA, and **Off-cap.** a contrasting adapter: `verification` on the localization and code-generation rows, `codegen_tooluse` on verification. Δ is On-cap. minus Mono. before rounding. All adapters are trained on Qwen2.5-Coder-7B-Instruct, and **Base** is the untrained model.

Probe	Instrument	Metric	On-cap.	Mono.	Off-cap.	Base	Δ (pp)
Loc.	TDD-Bench (Ahmed et al., 2024)	Test-file loc.	46.8	45.9	43.7	36.7	$+0.9$
Ver.	SWT-Bench (Mündler et al., 2024)	Fail-to-pass	6.2	6.2	3.5	6.0	$+0.0$
Code	SWE-bench oracle (Jimenez et al., 2024)	Resolved	11.8	13.2	7.0	6.2	-1.4

capability taxonomy, then test whether capability-masked adaptation retains competence on the assigned capability while inducing behavioral separation across specialists.

Capabilities in Agentic Coding Trajectories. An agentic coding episode on SWE-bench starts from a GitHub issue x and codebase c. The agent and repository environment alternately produce assistant turns and observations, so the interaction history can be written as

$$h_1 := (x, c), \qquad h_{t+1} := h_t \| (a_t, o_t), \tag{1}$$

where h_t is the history available before assistant turn t, a_t is that turn, o_t is the resulting environment observation, and $\|$ denotes ordered concatenation. A completed sequence is a trajectory τ, and $\mathcal{I}(\tau)$ indexes its assistant turns.

For our SWE instantiation, we study four capabilities that these turns exercise, written $\mathcal{A}$: `localization`, `code_generation`, `verification`, and `tool_use`. The capability taxonomy and the specialist parameterization are not one-to-one. In our trajectories, `tool_use` typically accompanies another capability and provides little standalone supervision. We therefore combine `code_generation` and `tool_use` into a single `codegen_tooluse` specialist, yielding three capability-specialized adapters: localization, verification, and code generation with tool use.

Capability masking retains comparable on-capability competence. To test whether capability-related competence can be localized to separate trainable modules, we introduce capability masking, detailed in Section 3.1. Each assistant turn is labeled by capability, and each specialist LoRA receives loss only on turns assigned to its capability set, while all other turns remain in context without contributing gradient. We compare the three masked specialist adapters against a monolithic LoRA trained on every turn and the untrained model, all built on the same frozen Qwen2.5-Coder-7B-Instruct base. We use three external probes targeting localization, verification, and code generation, each of which also evaluates an off-capability adapter as a contrast. Table 1 reports on-capability scores of 46.8 against the monolithic LoRA's 45.9 on TDD-Bench, 6.2 against 6.2 on SWT-Bench, and 11.8 against 13.2 on the SWE-bench oracle. The three deltas span -1.4 to $+0.9$ pp, showing that restricting gradient updates to capability-assigned turns retains performance close to whole-corpus monolithic adaptation on the corresponding probe, despite each specialist receiving supervision from only a subset of trajectory turns.

Capability masking induces behavioral separation across specialists. On TDD-Bench, the `localization` adapter exceeds the `verification` adapter by 3.1 pp. On the SWE-bench oracle, the `codegen_tooluse` adapter exceeds the `verification` adapter by 4.8 pp. On SWT-Bench, the off-capability `codegen_tooluse` adapter reaches 3.5%, below the 6.0–6.2% of the other arms. Patch application shows the same separation on the code-generation probe: 58.4% of the `codegen_tooluse` adapter's patches apply, against 51.4% for the monolithic LoRA, 38.2% for `verification`, 37.2% for `localization`, and 21.6% for the base model. These probe differences indicate that capability masking induces behaviorally differentiated specialists rather than replicas of the monolithic adapter.

Together, these results provide evidence for capability decoupling: masked adaptation retains comparable competence on the assigned capability while producing behaviorally differentiated, separately trainable parameter modules. Decoupling alone, however, does not yield a complete agent policy: long-horizon tasks require different capabilities to be invoked as the trajectory unfolds. TurnRoute therefore addresses the complementary problem of recomposition, selecting one specialist module for each turn during execution (Section 3).

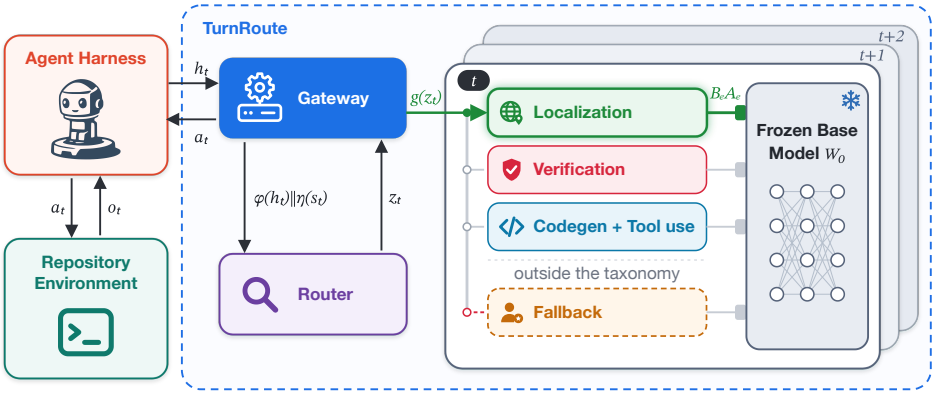


Figure 2: TurnRoute system overview for one assistant turn t. The unmodified agent harness sends the history h_t to the gateway, which passes the bounded rendering $\phi(h_t)\|\eta(s_t)$ to the router and receives a routing label z_t. The fixed map g dispatches z_t to a single adapter, whose trainable matrices B_e and A_e are applied over the frozen base model W_0; the fallback decision $\perp$ is mapped to the unmasked fallback adapter. The selected adapter generates the complete response a_t, which the harness executes in the repository environment to obtain o_t. The stacked panels denote later turns $t{+}1, t{+}2$, each routed afresh.

3 METHOD

Building on the capability-decoupling evidence above, TurnRoute addresses the complementary problem of recomposition. It trains capability-masked LoRA specialists over a shared frozen backbone and, before each assistant turn, routes generation to a single specialist while preserving the shared interaction history. TurnRoute therefore consists of two components: capability-masked expert training and turn-level routed composition. We further study two supervision schemes for learning the router, Hindsight and Foresight. Their details are elaborated in Sections 3.1–3.3, and Figure 2 gives an overview.

3.1 CAPABILITY-MASKED EXPERTS

Capability Taxonomy. TurnRoute uses the four recurring capabilities introduced in Section 2: `localization`, `code_generation`, `verification`, and `tool_use`. Two additional labels, `error` and `unknown`, cover turns outside these productive capabilities, forming the label space $\mathcal{B}$; only labels in $\mathcal{A}$ train specialists. Appendix A gives the formal definitions.

Trajectory Annotation. Let $\ell(\tau, t) \in \mathcal{B}$ denote the capability exercised by assistant turn t of a completed trajectory τ. A model-based annotator estimates it retrospectively. It receives the whole trajectory and labels every turn in one pass, so each turn is judged with its neighboring turns and observations in view. After annotation, every turn carries a single label $\widehat{\ell}(\tau, t) \in \mathcal{B}$. This label supplies both the training masks below and the Hindsight router targets of Section 3.3.

Capability Masking. The four capability labels induce three specialist target sets:

$$\begin{aligned} S_{\text{loc}} &= \{\texttt{localization}\}, \\ S_{\text{ver}} &= \{\texttt{verification}\}, \\ S_{\text{code}} &= \{\texttt{code_generation}, \texttt{tool_use}\}. \end{aligned} \tag{2}$$

A capability-masked adapter receives loss only on turns annotated with a capability in its target set $S_e \subseteq \mathcal{A}$:

$$m_e(\tau, t) := \mathbf{1}\left[\widehat{\ell}(\tau, t) \in S_e\right], \qquad t \in \mathcal{I}(\tau), \tag{3}$$

where $\mathbf{1}[\cdot]$ is the indicator function. Adapter e trains on turn t only when $m_e(\tau, t) = 1$, and masked turns stay in context without contributing gradient. The target sets do not overlap, so each turn trains at most one specialist. Setting $m_e \equiv 1$ everywhere gives the monolithic LoRA. Adapters couple only through the frozen base and the model interface, so one specialist adapter can be retrained or replaced without retraining the others.

Each adapter e is a pair of trainable matrices (B_e, A_e) (Hu et al., 2022) on top of frozen base parameters θ_0. Training minimizes the mean next-token cross-entropy over the unmasked assistant

tokens:

$$\Delta\theta_e^\star := \arg\min_{\Delta\theta} \mathcal{L}_e(\Delta\theta), \qquad \mathcal{L}_e(\Delta\theta) := \frac{-1}{N_e} \sum_{\tau,\, t\in\mathcal{I}(\tau)} m_e(\tau,t) \sum_{j=1}^{|a_t|} \log p_{\theta_0+\Delta\theta}(a_{t,j} \mid h_t, a_{t,<j}), \tag{4}$$

where $\Delta\theta$ ranges over candidate trainable matrices, the outer sum runs over training trajectories, and $N_e := \sum_{\tau,\, t\in\mathcal{I}(\tau)} m_e(\tau,t)\,|a_t|$ counts the unmasked assistant tokens. The minimizer $\Delta\theta_e^\star$ gives the trained matrices of adapter e.

The LoRA Adapter Pool. TurnRoute uses four adapters: three masked specialists for `localization`, `verification`, and `code_generation`+`tool_use`, plus one unmasked fallback adapter. We combine `tool_use` with `code_generation` because tool-use turns provide limited standalone supervision and primarily involve generating or executing scripts and commands. The fallback adapter is trained with $m_e \equiv 1$. The resulting pool is $\mathcal{P} := \{(B_e^\star, A_e^\star) : e \in \mathcal{E}\}$.

3.2 Turn-Level Routed Composition

Routing Mechanism. Before assistant turn t, after the preceding observation has entered h_t and before any token of a_t is generated, the router ρ emits one label from $\mathcal{Z} := \mathcal{A} \cup \{\perp\}$. The router reads text, not the raw history. Its input concatenates two renderings. $\phi(h_t)$ is a bounded textual rendering of the interaction history, which pins the task description and keeps the most recent messages within a fixed budget (Section 3.3). $\eta(s_t)$ is a textual rendering of the gateway state s_t, which records the most recently selected label and its consecutive run length. A fixed map $g : \mathcal{Z} \to \mathcal{E}$ sends `localization` and `verification` to their namesake adapters, sends `code_generation` and `tool_use` to `codegen_tooluse`, and maps $\perp$ to the unmasked `fallback` adapter. The selected adapter then generates the complete turn:

$$z_t = \rho(\phi(h_t) \| \eta(s_t)) \in \mathcal{Z}, \qquad e_t = g(z_t) \in \mathcal{E}, \qquad a_t \sim p_{\theta_0+\Delta\theta^\star_{e_t}}(\cdot \mid h_t), \tag{5}$$

where $\|$ is text concatenation, z_t is the selected label, e_t is the dispatched adapter, and $p_{\theta_0+\Delta\theta^\star_{e_t}}$ is the language-model distribution under that adapter. The router decides from the rendering, while the selected adapter conditions on the full history h_t. The fixed map gives every label a valid generation path, and the emitted label records which capability was selected. After generation, the environment returns o_t, and a_t and o_t enter h_{t+1} for the next decision. The adapters therefore act as sub-policies in sequential control rather than an ensemble over one output (Sutton et al., 1999). Appendix A gives the complete routing space.

Training and inference differ in the router input. At inference, $\eta(s_t)$ adds an advisory hint after four consecutive selections of the same label; both routers are trained on $\phi(h_t)$ alone, since their supervision states contain no such hints (Section 3.3).

Comparison with Token-Level Routing. A token-level gate mixes every expert's matrices at every token (Dou et al., 2024), whereas TurnRoute mounts one adapter's matrices at every adapted layer for the whole turn and therefore matches the activated adapter budget of a single monolithic LoRA during response generation. Appendix B writes out the served weights of both.

3.3 Router Supervision: Hindsight and Foresight

Bounded Router View. The router sees less context than the selected adapter. Its view pins the task description, excludes the system prompt, retains recent whole messages in chronological order, and drops the oldest messages when its budget is exceeded. Appendix B specifies the rendering budget, state update, and router-training objective.

Hindsight Supervision. Hindsight reuses the retrospective label of Section 3.1 as the router target $z_{\tau,t}^{\text{hind}}$ in Eq. (6), and a turn whose label names no capability is routed to `fallback`. It comes at no extra cost and yields 124,120 training states, each a truncated history h_t rendered as $\phi(h_t)$. The label was assigned after the fact, with the whole trajectory in view, including the turn a_t itself and its observation o_t. It therefore records what the rollout policy did at turn t, and a Hindsight router learns to reproduce that policy's sequence of capabilities, detours included.

Foresight Supervision. Foresight asks instead what should happen next. We sample 6,000 states uniformly from the same pool, truncate each before turn t, and show a prospective teacher T exactly

the bounded view that the router will see. The teacher names the capability the next, not yet written, step should exercise, giving the target $z^{\text{fore}}_{\tau,t}$ in Eq. (6), and never sees a_t. The two supervision schemes therefore differ primarily in decision perspective: retrospective behavior versus prospective action selection. The Hindsight target is constructed using information unavailable at routing time, including the realized turn and subsequent observation, whereas the Foresight target is defined using information available when the routing decision is made.

$$z^{\text{hind}}_{\tau,t} := \begin{cases} \widehat{\ell}(\tau,t), & \widehat{\ell}(\tau,t) \in \mathcal{A}, \\ \perp, & \text{otherwise}, \end{cases} \qquad z^{\text{fore}}_{\tau,t} := T(\phi(h_t)) \in \mathcal{Z}. \tag{6}$$

4 Experiments

4.1 Experimental Setup

Our experiments evaluate TurnRoute at two levels: capability probes of each adapter outside the agent loop, and end-to-end runs of the composed agent in the SWE-agent harness (Yang et al., 2024). The core configurations are summarized below, with further details in Appendix C.

Datasets. We train all adapters and routers on the SWE-smith trajectories that SWE-agent generated with Claude 3.7 Sonnet to train SWE-agent-LM-32B (Yang et al., 2025a). After screening and human evaluation by two annotators, we retain 4,837 of the 5,016 trajectories, about 124,000 assistant turns that Claude Sonnet 4.5 labels retrospectively. GPT-5.5 adds 6,000 prospective routing decisions. The probes of Table 1 evaluate localization on TDD-Bench Verified (Ahmed et al., 2024) by test-file localization, which credits a generated test patch that edits at least one file of the gold test patch, verification on SWT-Bench Verified (Mündler et al., 2024) by fail-to-pass rate, and code generation on SWE-bench Verified by resolution rate in the collapsed-oracle setting (SWE-bench oracle) of Jimenez et al. (2024), which provides the gold edit locations. End-to-end runs are evaluated on SWE-bench Verified (Chowdhury et al., 2024) and the SWE-bench Lite test split (Jimenez et al., 2024) by pass@1, at temperature 0 with a 75-step cap.

Baselines. Full fine-tuning and monolithic LoRA provide non-routed references, while single specialists and the random router test whether access to the specialist pool alone explains performance. LoRAMoE-A and LoRAMoE-B (Dou et al., 2024) provide token-level composition baselines at 7B and 14B, with ranks set to match TurnRoute's trained-parameter budget at each scale (Appendix C). All arms otherwise use the corresponding frozen backbone, training corpus, agent harness, and evaluation protocol.

4.2 Main Results

Table 2 reports results on Qwen2.5-Coder-7B-Instruct (Hui et al., 2024), where TurnRoute composes three rank-64 capability experts with one unmasked fallback adapter and every arm shares the base model, training corpus, agent harness, and benchmark. Tables 3 and 4 repeat the comparison on Qwen2.5-Coder-14B-Instruct and on SWE-bench Lite.

Dynamic recomposition recovers end-to-end competence. Despite decomposing adaptation across capability specialists, TurnRoute reaches 15.8% pass@1 at 7B, compared with 14.2% for monolithic LoRA and 16.2% for full fine-tuning (Table 2). At 14B, it reuses the 7B Foresight router without retraining and reaches 28.4%, compared with 26.4% and 31.0%, respectively (Table 3). The same ordering holds on SWE-bench Lite (Table 4).

Foresight routing supervision yields higher end-to-end performance. With the same 7B router backbone, Foresight reaches 15.8% pass@1 against 13.2% for Hindsight (Table 2). Foresight supervision also carries over to a smaller router from a different model family, as a Qwen3.5-2B Foresight router reaches 15.0%. Both Foresight routers achieve higher observed pass@1 than the monolithic LoRA, whereas Hindsight routing does not (13.2% against 14.2%), and a linear head on retrospective labels falls to 11.0%. Querying the teacher itself, GPT-5.5, as the router gives 17.0%. Given to the monolithic LoRA as an instruction, the Foresight decision alone reaches 16.8%.

Turn-level recomposition outperforms token-level composition at the same budget. With the trained-parameter budget matched, TurnRoute reaches 15.8% pass@1 at 7B against 9.2% for

Table 2: Main results on SWE-bench Verified with Qwen2.5-Coder-7B-Instruct. Resolved is the number of instances solved out of 500, and Δ is the pass@1 difference from monolithic LoRA in percentage points. Trainable parameters count locally trained expert and router weights. Learned routers are LoRA adapters on the same base model, except the 2B router, which uses Qwen3.5-2B (Qwen Team, 2026), and the linear head, a logistic regression on TF-IDF features. Capability-agnostic LoRA is a single LoRA trained on randomly chosen assistant turns that carry the same number of supervised tokens as the specialist, with no capability mask. †Foresight decisions are given to the monolithic LoRA as a one-line prompt instruction at every turn. Best result in bold.

Arm	Trainable params	Resolved	pass@1	Δ (pp)
Non-routed reference				
Full fine-tune	7.6B	81	16.2%	+2.0
Monolithic LoRA	161.5M	71	14.2%	0.0
Capability-agnostic LoRA	161.5M	70	14.0%	−0.2
TurnRoute (ours) — frontier API router				
GPT-5.5	4×161.5M	**85**	**17.0%**	+2.8
GPT-5.4-mini @high	4×161.5M	73	14.6%	+0.4
GPT-5.4-mini	4×161.5M	71	14.2%	0.0
TurnRoute (ours) — learned local router				
Foresight (7B)	4×161.5M + 161.5M	79	15.8%	+1.6
Foresight (2B)	4×161.5M + 62.5M	75	15.0%	+0.8
Hindsight (7B)	4×161.5M + 161.5M	66	13.2%	−1.0
Linear head	4×161.5M + 0.25M	55	11.0%	−3.2
Matched-budget token-level composition baseline				
LoRAMoE-A	649.4M	46	9.2%	−5.0
LoRAMoE-B	646.3M	43	8.6%	−5.6
Routing signal without specialists				
Foresight (7B) + Monolithic LoRA†	161.5M + 161.5M	84	16.8%	+2.6
Specialists without routing signal				
Random router	3×161.5M	44	8.8%	−5.4
Single-adapter ablations (no router)				
Localization	161.5M	45	9.0%	−5.2
Codegen+tool-use	161.5M	38	7.6%	−6.6
Verification	161.5M	0	0.0%	−14.2

LoRAMoE-A and 8.6% for LoRAMoE-B (Table 2), and 28.4% at 14B against 14.2% and 13.4% (Table 3). Both LoRAMoE variants fall below the monolithic LoRA at both scales, so spending the same budget on per-token gates does not reproduce what turn-level selection achieves. TurnRoute is also cheaper to serve. It activates one adapter per turn, whereas LoRAMoE runs all of its experts on every token, and its adapters run on standard multi-LoRA serving, whereas LoRAMoE's input-dependent gate cannot be merged into the base weights (Appendix B).

TurnRoute raises patch production. At 7B, Foresight produces a non-empty patch on 94.4% of Verified instances, against 79.8% for full fine-tuning and 77.8% for monolithic LoRA, removing 74.8% of the baseline's empty-patch failures, and at 14B it reaches 97.4% (Figure 3). Both LoRAMoE variants invert the pattern at both scales, producing a patch on 43.4% and 44.6% of instances at 7B and on 62.2% and 54.0% at 14B, against 77.8% and 85.2% for the monolithic LoRA, and resolving between 19.3% and 24.8% of what they produce. The dominant behavioral effect of TurnRoute is therefore on whether an episode reaches a submitted edit.

5 Analysis

Capability Specialist Dissociation. The probes of Section 2 show that capability masking produces behaviorally differentiated specialists. The end-to-end ablation asks whether any one of these specialists nevertheless retains a complete agent policy. Served alone without a router, every specialist falls well below the monolithic baseline trained on the same corpus: `localization` reaches 9.0% pass@1, `codegen_tooluse` 7.6%, and `verification` resolves nothing (Table 2). The specialists therefore behave as partial policies rather than replicas of the monolithic agent, making recomposition at serving time a necessary part of TurnRoute.

Table 3: Results on SWE-bench Verified (500 instances) with Qwen2.5-Coder-14B-Instruct. Δ is the pass@1 difference from monolithic LoRA in percentage points. TurnRoute reuses the Foresight (7B) router of Table 2 without retraining. Best result in bold, second best underlined.

Arm	pass@1	Δ (pp)
Non-routed reference		
Full fine-tune	**31.0%**	+4.6
Monolithic LoRA	26.4%	0.0
TurnRoute (ours) — learned local router		
Foresight (7B)	<u>28.4%</u>	+2.0
Matched-budget token-level composition baseline		
LoRAMoE-A	14.2%	−12.2
LoRAMoE-B	13.4%	−13.0

Table 4: Results on SWE-bench Lite (300 instances). Δ is the pass@1 difference from monolithic LoRA at each scale in percentage points. TurnRoute uses the Foresight (7B) router at both scales, without retraining. Best result per scale in bold, second best underlined.

Arm	pass@1	Δ (pp)
7B		
Full fine-tune	**13.7%**	+2.0
Monolithic LoRA	11.7%	0.0
TurnRoute, Foresight (7B)	<u>12.3%</u>	+0.7
14B		
Full fine-tune	**26.7%**	+5.3
Monolithic LoRA	21.3%	0.0
TurnRoute, Foresight (7B)	<u>24.0%</u>	+2.7

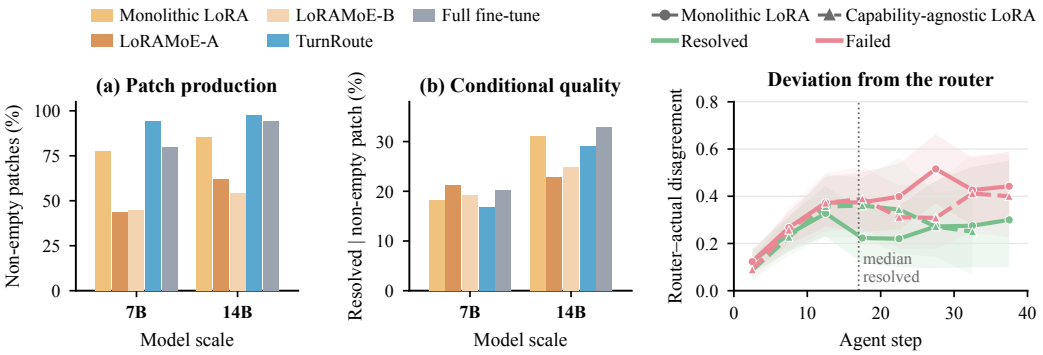


Figure 3: Patch-production decomposition on SWE-bench Verified for matched end-to-end arms with Qwen2.5-Coder-7B-Instruct and Qwen2.5-Coder-14B-Instruct. Panel (a) reports the share of instances yielding a non-empty patch, and panel (b) the resolution rate among those instances.

Figure 4: Disagreement between the Foresight router's prediction and the agent's step function, labeled by GPT-5.5. Episodes come from 50 SWE-bench Verified instances.

Supervision-Budget Control. The capability-agnostic LoRA is trained on a random 27% of the supervised tokens, matching the `codegen_tooluse` specialist's budget without a capability mask. It reaches 14.0% pass@1, close to monolithic LoRA (14.2%), whereas the masked specialist reaches 7.6% (Table 2). With the probes of Section 2, this indicates that masking decomposes the policy rather than producing a weaker copy of the monolithic agent.

Routing-Signal Ablation. The two routing controls in Table 2 each isolate one ingredient of TurnRoute. The random router keeps the three specialists but picks one uniformly at random for each turn, so the agent still switches between specialists without matching them to the capability the turn needs. It falls to 8.8% pass@1, 5.4 pp below monolithic LoRA. Conversely, passing the Foresight decision to the monolithic LoRA as a one-line prompt keeps the signal but drops the specialists, and reaches 16.8%, against 15.8% for TurnRoute. Most of the gain therefore comes from the per-turn routing signal. Thus, modularization preserves competence but is not itself the sole source of the pass@1 gain.

Deviation from Capability-Aligned Behavior. To locate where failing episodes go wrong, we compare failed and resolved episodes of the same issue. We run the 7B monolithic and capability-agnostic LoRA on 50 instances, label the function of every step with GPT-5.5, and compare it with the capability the Foresight router predicts for that step. On 29 instances, one adapter resolves the issue and the other fails. In 21 of them, the failed episode disagrees with the router on a larger share of its steps than the resolved one. Both episodes follow the router equally closely during early exploration, but failing episodes keep deviating later (Figure 4).

6 Related Work

LLM Agents for Interactive Coding. SWE-bench evaluates end-to-end repository repair from real GitHub issues (Jimenez et al., 2024), with SWE-bench Verified providing a human-validated subset (Chowdhury et al., 2024). Interactive agents such as SWE-agent interleave reasoning, action, and environment observation over long trajectories (Yao et al., 2023; Yang et al., 2024), while SWE-Gym and SWE-smith provide executed trajectories for training (Pan et al., 2025; Yang et al., 2025a). Complementary benchmarks isolate capabilities such as localization and verification (Chen et al., 2025; Ahmed et al., 2024; Mündler et al., 2024). Together, these resources make software engineering a natural setting for studying whether recurring agent capabilities can be learned and composed modularly.

Capability Decomposition for Coding Agents. Agentless fixes a localization–repair–validation pipeline and MASAI delegates reproduction, localization, fixing, and ranking to specialized sub-agents (Xia et al., 2024; Arora et al., 2024). SWE-Fixer separates retrieval and editing, Kimi-Dev injects stage-specific priors from annotated trajectories, and AgentLens labels turn-level intent for diagnosis (Xie et al., 2025; Yang et al., 2025b; Sahoo et al., 2026). These methods introduce structure through prompts, pipelines, stage-specific training, or separate models, but do not decompose recurring capabilities within a single agent policy into independently trainable modules that are dynamically recomposed during execution. TRACE comes closest to capability-level modular training: it identifies missing capabilities from successful and failed trajectories, synthesizes new environments, and trains dedicated LoRA experts with reinforcement learning (Kang et al., 2026). TurnRoute instead reorganizes capabilities already present in a fixed corpus of agent trajectories, learning capability-specialized modules without collecting new environments, rollouts, or rewards, and recomposing them turn by turn within the same trajectory.

LoRA Composition, Routing, and Serving. LoRA provides a lightweight trainable module over a shared backbone (Hu et al., 2022). TIES and DARE construct fixed merged updates, while LoRAHub searches composition weights over an adapter library (Yadav et al., 2023; Yu et al., 2024; Huang et al., 2023); these compositions are determined before generation. LoRAMoE, PHATGOOSE, Arrow, X-LoRA, LoRA-Flow, and MixLoRA make adapter use input-dependent (Dou et al., 2024; Muqeeth et al., 2024; Ostapenko et al., 2024; Buehler & Buehler, 2024; Wang et al., 2024; Li et al., 2024), but route at the token, layer, or module level, below the granularity of an observable agent action. DialRouter and MTRouter condition on dialogue history but select whole models (Zhang et al., 2026a;b), while Skill-to-LoRA and Adaptive Minds expose adapters as loadable skills or invocable tools (Zhang & Qi, 2026; Shekar & Krishnan, 2025). TurnRoute differs in both how experts are formed and how they are composed: specialists are trained by capability-masking turns from existing agent trajectories, and one specialist is selected for an entire assistant turn. Thus, routing corresponds to an inspectable capability-level decision and composition occurs at the granularity of agent behavior rather than individual tokens or hidden modules. Advances in multi-adapter serving (Jeon et al., 2026; Li et al., 2025; Woo et al., 2026; Wang et al., 2026) are orthogonal to TurnRoute and can support its deployment.

7 Conclusion

Findings. We introduced TurnRoute, a framework for separating recurring agent capabilities into independently trained parameter modules and dynamically recomposing them at the turn level. Capability masking produces behaviorally differentiated specialists that retain comparable competence on their assigned capability but are insufficient as standalone agents. Their dynamic recomposition recovers end-to-end competence and achieves higher observed pass@1 than monolithic LoRA at both 7B and 14B, with the clearest behavioral effect being a substantial reduction in empty-patch failures. We further find that prospective Foresight supervision is more effective than reproducing the capability sequence of the original rollout, while much of the end-to-end gain comes from this routing signal itself. **Limitations and Scope.** Our evidence is limited to software-engineering agents, a capability taxonomy we defined, and single-run evaluations. Foresight and Hindsight also differ in supervision construction, so their performance gap does not isolate decision perspective alone. TurnRoute also trades additional training and resident storage for modularity; we therefore view the results as evidence of feasibility rather than universal optimality.

AI USE STATEMENT

In this work, we used generative AI tools for drafting and refining the writing, for coding assistance, and for searching the literature and formatting references. We used LLMs for automated trajectory annotation and routing-label generation, as described in the methodology. The authors reviewed and take responsibility for the research design, experiments, analysis, claims, and final presentation.

ETHICS STATEMENT

This work uses only public resources: the SWE-smith trajectory corpus and SWE-bench Verified and Lite, which are built from public GitHub repositories under their original licenses. It involves no human subjects or personal data. Capability and routing labels were produced by commercial language-model APIs under their terms of use, and human annotators reviewed the capability labels. Two annotators also evaluated the trajectories during corpus screening. The reviewed material consists only of code and agent interaction logs, so the review exposed annotators to no personal or harmful content. Every agent episode ran in an isolated per-instance container, and no patch was submitted to an upstream repository.

REPRODUCIBILITY STATEMENT

Sections 4.1 and 4.2 describe the base models, the trajectory corpus and its capability annotation, the agent harness and interaction budget, the grading protocol, and the benchmark splits. The capability taxonomy is in Appendix A, and the router's input rendering, gateway state, and training objective are in Appendix B. The full training configuration of the experts, the full fine-tune, the routers, and the LoRAMoE baselines is in Appendix C, and parameter and memory accounting is in Appendix E. All end-to-end results are graded with the official SWE-bench harness. Each arm is a single run, so claimed differences in patch production are tested with exact paired McNemar tests (Appendix D).

REFERENCES

Toufique Ahmed, Martin Hirzel, Rangeet Pan, Avraham Shinnar, and Saurabh Sinha. TDD-Bench Verified: Can LLMs generate tests for issues before they get resolved? *CoRR*, abs/2412.02883, 2024. doi: 10.48550/ARXIV.2412.02883. URL https://doi.org/10.48550/arXiv.2412.02883.

Daman Arora, Atharv Sonwane, Nalin Wadhwa, Abhav Mehrotra, Saiteja Utpala, Ramakrishna Bairi, Aditya Kanade, and Nagarajan Natarajan. MASAI: Modular architecture for software-engineering AI agents. *CoRR*, abs/2406.11638, 2024. doi: 10.48550/ARXIV.2406.11638. URL https://doi.org/10.48550/arXiv.2406.11638.

Eric L. Buehler and Markus J. Buehler. X-LoRA: Mixture of low-rank adapter experts, a flexible framework for large language models with applications in protein mechanics and molecular design. *APL Machine Learning*, 2(2):026119, 2024. doi: 10.1063/5.0203126. URL https://arxiv.org/abs/2402.07148.

Dong Chen, Shaoxin Lin, Muhan Zeng, Daoguang Zan, Jian-Gang Wang, Anton Cheshkov, Jun Sun, Hao Yu, Guoliang Dong, Artem Aliev, Jie Wang, Xiao Cheng, Guangtai Liang, Yuchi Ma, Pan Bian, Tao Xie, and Qianxiang Wang. CodeR: Issue resolving with multi-agent and task graphs. *CoRR*, abs/2406.01304, 2024. doi: 10.48550/ARXIV.2406.01304. URL https://doi.org/10.48550/arXiv.2406.01304.

Zhaoling Chen, Robert Tang, Gangda Deng, Fang Wu, Jialong Wu, Zhiwei Jiang, Viktor K. Prasanna, Arman Cohan, and Xingyao Wang. LocAgent: Graph-guided LLM agents for code localization. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 8697–8727. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.ACL-LONG.426. URL https://doi.org/10.18653/v1/2025.acl-long.426.

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench Verified. OpenAI, https://openai.com/index/introducing-swe-bench-verified/, 2024. Accessed August 2026.

Shihan Dou, Enyu Zhou, Yan Liu, Songyang Gao, Wei Shen, Limao Xiong, Yuhao Zhou, Xiao Wang, Zhiheng Xi, Xiaoran Fan, Shiliang Pu, Jiang Zhu, Rui Zheng, Tao Gui, Qi Zhang, and Xuanjing Huang. LoRAMoE: Alleviating world knowledge forgetting in large language models via MoE-style plugin. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 1932–1945. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.106. URL https://doi.org/10.18653/v1/2024.acl-long.106.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. LoraHub: Efficient cross-task generalization via dynamic LoRA composition. *CoRR*, abs/2307.13269, 2023. doi: 10.48550/ARXIV.2307.13269. URL https://doi.org/10.48550/arXiv.2307.13269.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-Coder technical report. *CoRR*, abs/2409.12186, 2024. doi: 10.48550/ARXIV.2409.12186. URL https://doi.org/10.48550/arXiv.2409.12186.

Hyesung Jeon, Hyeongju Ha, and Jae-Joon Kim. LRAgent: Efficient KV cache sharing for multi-LoRA LLM agents. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=ooRukY3zjQ.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world Github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Hangoo Kang, Tarun Suresh, Jon Saad-Falcon, and Azalia Mirhoseini. TRACE: capability-targeted agentic training. *CoRR*, abs/2604.05336, 2026. doi: 10.48550/ARXIV.2604.05336. URL https://doi.org/10.48550/arXiv.2604.05336.

Allison Li, Kristjan Greenewald, Thomas Parnell, and Navid Azizan. Efficient multi-adapter LLM serving via cross-model KV-cache reuse with activated LoRA. *CoRR*, abs/2512.17910, 2025. doi: 10.48550/ARXIV.2512.17910. URL https://doi.org/10.48550/arXiv.2512.17910.

Dengchun Li, Yingzi Ma, Naizheng Wang, Zhengmao Ye, Zhiyuan Cheng, Yinghao Tang, Yan Zhang, Lei Duan, Jie Zuo, Cal Yang, and Mingjie Tang. MixLoRA: Enhancing large language models fine-tuning with LoRA-based mixture of experts. *CoRR*, abs/2404.15159, 2024. doi: 10.48550/ARXIV.2404.15159. URL https://doi.org/10.48550/arXiv.2404.15159.

Niels Mündler, Mark Niklas Müller, Jingxuan He, and Martin Vechev. SWT-Bench: Testing and validating real-world bug-fixes with code agents. In Amir Globerson, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, pp. 81857–81887, 2024. doi: 10.52202/

079017-2601. URL https://papers.nips.cc/paper_files/paper/2024/hash/94f093b41fc2666376fb1f667fe282f3-Abstract-Conference.html.

Mohammed Muqeeth, Haokun Liu, Yufan Liu, and Colin Raffel. Learning to route among specialized experts for zero-shot generalization. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 36829–36846. PMLR, 2024. URL https://proceedings.mlr.press/v235/muqeeth24a.html.

Oleksiy Ostapenko, Zhan Su, Edoardo M. Ponti, Laurent Charlin, Nicolas Le Roux, Lucas Caccia, and Alessandro Sordoni. Towards modular LLMs by building and reusing a library of LoRAs. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 38885–38904. PMLR, 2024. URL https://proceedings.mlr.press/v235/ostapenko24a.html.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*, pp. 47717–47737. PMLR, 2025. URL https://proceedings.mlr.press/v267/pan25g.html.

Qwen Team. Qwen3.5-2B. Hugging Face model hub, https://huggingface.co/Qwen/Qwen3.5-2B, 2026. Model card for the released Qwen3.5-2B checkpoint used for the router scale-down ablation; accessed August 2026.

Priyam Sahoo, Gaurav Mittal, Xiaomin Li, Shengjie Ma, Benjamin Steenhoek, Pingping Lin, and Yu Hu. AgentLens: Revealing the lucky pass problem in SWE-Agent evaluation. *CoRR*, abs/2605.12925, 2026. doi: 10.48550/ARXIV.2605.12925. URL https://doi.org/10.48550/arXiv.2605.12925.

Pavan C. Shekar and Aswanth Krishnan. Adaptive Minds: Empowering agents with LoRA-as-tools. *CoRR*, abs/2510.15416, 2025. doi: 10.48550/ARXIV.2510.15416. URL https://doi.org/10.48550/arXiv.2510.15416.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, 1999. doi: 10.1016/S0004-3702(99)00052-1. URL https://doi.org/10.1016/S0004-3702(99)00052-1.

Trae Research Team, Pengfei Gao, Zhao Tian, Xiangxin Meng, Xinchen Wang, Ruida Hu, Yuanan Xiao, Yizhou Liu, Zhao Zhang, Junjie Chen, Cuiyun Gao, Yun Lin, Yingfei Xiong, Chao Peng, and Xia Liu. Trae Agent: An LLM-based agent for software engineering with test-time scaling. *CoRR*, abs/2507.23370, 2025. doi: 10.48550/ARXIV.2507.23370. URL https://doi.org/10.48550/arXiv.2507.23370.

Hanqing Wang, Bowen Ping, Shuo Wang, Xu Han, Yun Chen, Zhiyuan Liu, and Maosong Sun. LoRA-Flow: Dynamic LoRA fusion for large language models in generative tasks. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 12871–12882. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.695. URL https://doi.org/10.18653/v1/2024.acl-long.695.

Shao Wang, Rui Ren, and Lin Gui. ForkKV: Scaling multi-LoRA agent serving via copy-on-write disaggregated KV cache. *CoRR*, abs/2604.06370, 2026. doi: 10.48550/ARXIV.2604.06370. URL https://doi.org/10.48550/arXiv.2604.06370.

Sunghyeon Woo, Jaeeun Kil, Hoseung Kim, Minsub Kim, Joonghoon Kim, Ahreum Seo, Sungjae Lee, Minjung Jo, Jiwon Ryu, Baeseong Park, Se Jung Kwon, and Dongsoo Lee. ICaRus: Identical cache reuse for efficient multi model inference. *CoRR*, abs/2603.13281, 2026. doi: 10.48550/ARXIV.2603.13281. URL https://doi.org/10.48550/arXiv.2603.13281.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based software engineering agents. *CoRR*, abs/2407.01489, 2024. doi: 10.48550/ARXIV.2407.01489. URL https://doi.org/10.48550/arXiv.2407.01489.

Chengxing Xie, Bowen Li, Chang Gao, He Du, Wai Lam, Difan Zou, and Kai Chen. SWE-Fixer: Training open-source LLMs for effective and efficient GitHub issue resolution. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, volume ACL 2025 of *Findings of ACL*, pp. 1123–1139. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-ACL.62. URL https://doi.org/10.18653/v1/2025.findings-acl.62.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A. Raffel, and Mohit Bansal. TIES-Merging: Resolving interference when merging models. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, pp. 7093–7115, 2023. doi: 10.52202/075280-0310. URL https://papers.nips.cc/paper_files/paper/2023/hash/1644c9af28ab7916874f6fd6228a9bcf-Abstract-Conference.html.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In Amir Globerson, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, pp. 50528–50652, 2024. doi: 10.52202/079017-1601. URL https://papers.nips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, pp. 107402–107452, 2025a. doi: 10.52202/085713-3239. URL https://papers.nips.cc/paper_files/paper/2025/hash/8b86cf5ace600c48fd188efbb8dedec8-Abstract-Datasets_and_Benchmarks_Track.html.

Zonghan Yang, Shengjie Wang, Kelin Fu, Wenyang He, Weimin Xiong, Yibo Liu, Yibo Miao, Bofei Gao, Yejie Wang, Yingwei Ma, Yanhao Li, Yue Liu, Zhenxing Hu, Kaitai Zhang, Shuyi Wang, Huarong Chen, Flood Sung, Yang Liu, Yang Gao, Zhilin Yang, and Tianyu Liu. Kimi-Dev: Agentless training as skill prior for SWE-Agents. *CoRR*, abs/2509.23045, 2025b. doi: 10.48550/ARXIV.2509.23045. URL https://doi.org/10.48550/arXiv.2509.23045.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are Super Mario: Absorbing abilities from homologous models as a free lunch. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.),

Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024, volume 235 of *Proceedings of Machine Learning Research*, pp. 57755–57775. PMLR, 2024. URL https://proceedings.mlr.press/v235/yu24p.html.

Jiarui Zhang, Xiangyu Liu, Yong Hu, Chaoyue Niu, Hang Zeng, Shaojie Tang, Fan Wu, and Guihai Chen. From myopic selection to long-horizon awareness: Sequential LLM routing for multi-turn dialogue. *CoRR*, abs/2604.12385, 2026a. doi: 10.48550/ARXIV.2604.12385. URL https://doi.org/10.48550/arXiv.2604.12385.

Tianyi Zhang and Zhonghao Qi. Skill-to-LoRA: From using skills to learning behaviors for token-efficient LLM agents. *CoRR*, abs/2606.16769, 2026. doi: 10.48550/ARXIV.2606.16769. URL https://doi.org/10.48550/arXiv.2606.16769.

Yiqun Zhang, Hao Li, Zihan Wang, Shi Feng, Xiaocui Yang, Daling Wang, Bo Zhang, Lei Bai, and Shuyue Hu. MTRouter: Cost-aware multi-turn LLM routing with history-model joint embeddings. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 44206–44226. Association for Computational Linguistics, 2026b. doi: 10.18653/V1/2026.ACL-LONG.2045.

A CAPABILITY TAXONOMY

Table 5 defines each capability by what a step consumes and what it produces. We write $\mathcal{A}$ for the four productive capabilities and $\mathcal{B} := \mathcal{A} \cup \{\texttt{error}, \texttt{unknown}\}$ for the full label space. Each turn's retrospective annotation is a single label in $\mathcal{B}$, the first capability the annotator lists. The router predicts from $\mathcal{Z} := \mathcal{A} \cup \{\perp\}$, where $\perp$ is emitted as the `unknown` label and sends the turn to the `fallback` adapter (Section 3.2). $\mathcal{Z}$ omits `error`, which describes a completed step and has no next-turn counterpart.

Table 5: The capability taxonomy. Each productive capability is defined by what a step consumes and what it produces. `error` and `unknown` complete the label space.

Capability	Role of the step	Consumes	Produces
`localization`	Locate the target functions and files	Issue description, code structure	Candidate files and functions
`code_generation`	Produce the patch that fixes the described problem	Issue description, candidate files	Modified source files
`verification`	Produce or run tests that check whether the patch fixes the problem	Modified source files	Test outcome for the patch
`tool_use`	Produce and run auxiliary scripts supporting the workflow, such as shell or version-control operations	Workflow state	New tool scripts and/or effects in the environment
`error`	Step is unnecessary or wrong for the task	—	—
`unknown`	Step exercises none of the above and is not an error	—	—

B ROUTER INPUT, STATE AND OBJECTIVE

Recall from Section 3.2 that, before assistant turn t, the router ρ reads a rendering of the history h_t and of the gateway state s_t, emits a label z_t, and the fixed map g dispatches the turn to a single adapter e_t, which generates it from the full history:

$$z_t = \rho(\phi(h_t) \| \eta(s_t)) \in \mathcal{Z}, \qquad e_t = g(z_t) \in \mathcal{E}, \qquad a_t \sim p_{\theta_0 + \Delta\theta^\star_{e_t}}(\cdot \mid h_t).$$

This appendix specifies the history rendering ϕ, the state s_t and its rendering η, and the objective that trains ρ.

Bounded Router View. The bounded router view $\phi(h_t)$ pins the task description, clipped to 2,500 characters, and excludes the agent system prompt. It retains at most the six most recent messages, each clipped to 1,200 characters. Whole messages are dropped oldest-first until the serialization fits within a 9,000-character budget, and the retained messages remain in chronological order.

Gateway State. Let $s_t = (\bar{z}_t, c_t)$ record the most recently selected label $\bar{z}_t$, that is z_{t-1} for $t > 1$, and its consecutive selection count c_t before turn t. The state is initialized and updated as

$$s_1 := (\perp, 0), \qquad s_{t+1} := \begin{cases} (z_t,\ c_t + 1), & z_t = \bar{z}_t, \\ (z_t,\ 1), & z_t \neq \bar{z}_t. \end{cases} \tag{7}$$

The rendering $\eta(s_t)$ is empty unless $c_t \geq K$ with $K = 4$, in which case it is an advisory streak hint asking the router to reconsider.

Router Objective. The supervision states carry no streak hint, so a router with parameters ψ is trained on the history rendering alone. It minimizes the target cross-entropy over a supervision set $\mathcal{D}$ of (history, target) pairs, where the target z^* is $z^{\text{hind}}_{\tau,t}$ for the Hindsight router and $z^{\text{fore}}_{\tau,t}$ for the Foresight router (Eq. (6)):

$$\mathcal{L}_\rho(\psi) = -\sum_{(h,z^*)\in\mathcal{D}} \log q_\psi(z^* \mid \phi(h)), \qquad \psi^\star := \arg\min_\psi \mathcal{L}_\rho(\psi), \tag{8}$$

where q_ψ is the predictive distribution over $\mathcal{Z}$ and the loss covers only the tokens of the capability label.

Training Versus Inference. The router input therefore differs between the two phases. During training it is always $\phi(h)$. At inference it is $\phi(h_t)\|\eta(s_t)$, and $\eta(s_t)$ is empty, so the input reduces to the training form $\phi(h_t)$, until the same label has been selected $K = 4$ times in a row. Only on those turns does the gateway append the streak hint, an input the router never saw during training. The hint is advisory: the router may keep the current label, and the dispatch map g is unchanged. The streak hint originates in our first end-to-end runs, which queried GPT-5.5 directly as the router (Table 2), where it guards against the router repeating one label indefinitely. We keep the same inference pipeline, hint included, for every router we evaluate, so that trained and queried routers differ only in the router itself.

Comparison with Token-Level Routing. TurnRoute fixes the served weight for the whole turn. Writing $(B^{(t)}, A^{(t)}) := (B^\star_{g(z_t)}, A^\star_{g(z_t)}) \in \mathcal{P}$, every token of turn t is generated under $W^{(t)} = W_0 + B^{(t)}A^{(t)}$ at every adapted layer. LoRAMoE (Dou et al., 2024), our token-level baseline, instead attaches N LoRA experts to each feed-forward projection and mixes them per token with a softmax gate trained jointly with the experts. Omitting the LoRA scaling factor in both cases, the weight used for token j of turn t is

$$W^{(t,j)}_{\text{mix}} = W_0 + \sum_{k=1}^{N} \pi_k(v_{t,j})\, B_k A_k, \qquad \pi(v_{t,j}) = \text{softmax}(W_g v_{t,j})\,, \tag{9}$$

where $v_{t,j}$ is the token's hidden state at that layer and W_g is the gate. Every expert runs on every token, so LoRAMoE activates all N experts' matrices at each step, whereas TurnRoute mounts one adapter per turn and keeps the activated budget of a single LoRA. The gate also decides per token and per layer, below the level of an agent action, while TurnRoute makes one capability decision per turn.

C Training Hyperparameters

Expert Training. All LoRA adapters use rank 64 and $\alpha = 128$ on the attention query, key, value, and output projections and on the MLP projections, with the output head frozen. Training uses seed 42, shuffled data, bfloat16, activation checkpointing, chunked cross-entropy, fused AdamW, and a cosine schedule with five warm-up steps. Each run lasts three epochs at a 32,768-token sequence length and excludes input tokens from the loss. Table 6 lists the settings that differ across arms.

Table 6: Adapter and full fine-tuning hyperparameters for Qwen2.5-Coder-7B-Instruct (7B) and Qwen2.5-Coder-14B-Instruct (14B). Effective batch size is measured across all ranks.

Model and arm	Type	Learning rate	Weight decay	Effective batch
Full fine-tune (7B)	Full	1×10^{-4}	0.01	32
Monolithic LoRA (7B)	LoRA	2×10^{-4}	0.01	16
Three capability experts (7B)	LoRA	2×10^{-4}	0.01	16
Full fine-tune (14B)	Full	1×10^{-4}	0.01	32
Monolithic LoRA and experts (14B)	LoRA	2×10^{-4}	0.01	16

Training ran on GH200 nodes of a national HPC service. Each 14B job used 16 ranks on four nodes and took 4.5–4.8 hours. A 14B adapter occupies 525 MB, against 28 GB for a full fine-tuning checkpoint.

Router Training. The neural routers use $\alpha = 128$, a learning rate of 2×10^{-4}, 8,192-token sequences, and an effective batch size of 128, with the loss restricted to the capability-tag span (Table 7).

The linear router is a logistic regression over the five labels of $\mathcal{Z}$, on TF–IDF unigram and bigram features capped at 50,000. It has $5 \times 50{,}000$ weights and five intercepts, or 250,005 parameters. It

takes 4 MB and runs on CPU in about 6 ms per call. Foresight-2B replaces the 7B router backbone with Qwen3.5-2B (Qwen Team, 2026) and keeps the Qwen2.5-Coder-7B experts, the Foresight training set, the hyperparameters, and the interface.

Table 7: Router supervision and hyperparameters. Hindsight and Foresight are rank-64 LoRA adapters on Qwen2.5-Coder-7B-Instruct, and Foresight-2B on Qwen3.5-2B. Example counts are shown as training/validation.

Router	Supervision	Examples	Epochs
Hindsight	Retrospective model tags	124,120 / 5,927	1
Foresight	Prospective teacher decisions	6,000 / 800	4
Foresight-2B	Prospective teacher decisions	6,000 / 800	4

LoRAMoE Baselines. Both LoRAMoE variants follow Dou et al. (2024). Each attaches N LoRA experts to the gate, up, and down projections of every feed-forward block and mixes them per token with a softmax gate trained jointly with the experts. LoRAMoE-B adds the published localized balancing constraint, which splits the experts into two equal groups and assigns each training trajectory to one group according to whether at least half of its assistant turns are labeled `localization`. The reference implementation supports only two expert groups and one binary label per sample, so our per-turn capability labels must be reduced to a single split. We split on `localization`, which labels 35.1% of assistant turns and at least half of the turns in 771 of the 4,837 trajectories. Ranks are set so that the N experts together match TurnRoute's trained-parameter budget at each scale, four rank-64 adapters of 161.5M parameters at 7B and of 275.3M at 14B. Table 8 lists the full configuration.

Table 8: LoRAMoE baseline hyperparameters. Values spanning both columns are shared by the two variants and both model scales.

Hyperparameter	LoRAMoE-A	LoRAMoE-B
Experts N	3	4
Rank (7B / 14B)	114 / 135	85 / 101
Balancing weight / α	—	0.1 / 0.1
Adapted projections	FFN gate / up / down	
LoRA α / dropout	32 / 0.05	
Optimizer	AdamW	
Learning rate	2×10^{-4}	
Weight decay	0	
Schedule	Cosine (3% warm-up)	
Gradient clipping	1.0	
Effective batch	8	
Epochs	3	
Sequence length	32,768	
Precision	bfloat16	
Seed	41	

D Patch Production on SWE-bench Lite

Table 9 gives pass@1 and the empty-patch rate on the 300 SWE-bench Lite test instances for the arms of Table 4. Table 10 tests the patch-production differences between TurnRoute and monolithic LoRA on Lite and on the Verified results of Section 4 with an exact two-sided McNemar test over instances attempted by both arms.

E Parameter and Memory Accounting

At 14B, the safetensors headers give 14.770B base parameters and 275.3M parameters per rank-64 adapter, and Table 11 lists the resulting totals.

Table 9: Pass@1 and empty-patch rate on the 300 SWE-bench Lite test instances for the arms of Table 4.

Arm	pass@1	Empty patch
7B		
Full fine-tune	13.7%	19.3%
Monolithic LoRA (baseline)	11.7%	23.7%
TurnRoute (Foresight)	12.3%	4.3%
14B		
Full fine-tune	26.7%	8.3%
Monolithic LoRA (baseline)	21.3%	13.0%
TurnRoute (Foresight)	24.0%	1.0%

Table 10: Exact paired McNemar tests on patch production. b counts instances on which the first arm produced a non-empty patch and the second did not, and c counts the reverse. Verified-500 rows use the production definition of Figure 3, which also counts an unsubmitted instance as producing no patch.

Comparison	Instance set	b	c	p
TurnRoute vs. monolithic LoRA (7B)	Lite-300	63	5	7.7×10^{-14}
TurnRoute vs. monolithic LoRA (7B)	Verified-500	90	7	1.8×10^{-19}
TurnRoute vs. monolithic LoRA (14B)	Verified-500	67	4	8.7×10^{-16}

Table 11: Parameter counts for one complete Qwen2.5-Coder-14B-Instruct system. Activated parameters are counted per generated token, with one TurnRoute expert mounted at a time.

System	Trained	Activated	Relative trained
Full fine-tune	14.770B	14.770B	$1\times$
Monolithic LoRA	0.275B	15.045B	$53.7\times$ fewer
TurnRoute, four adapters	1.101B	15.045B	$13.4\times$ fewer
TurnRoute with router	1.263B	15.045B	$11.7\times$ fewer

A monolithic LoRA and a TurnRoute turn both activate 15.045B parameters per generated token, since TurnRoute mounts one adapter at a time. TurnRoute adds a 7B LoRA router call of 0.37 s per turn. Serving one base, four adapters, and the router needs at least 47.3 GB, against 30.1 GB for the monolithic LoRA, or $1.6\times$ the resident memory. Training the four adapters costs nearly four monolithic training runs and $1.95\times$ the supervised tokens.

F Routing Dynamics

This appendix shows how the Foresight router uses the experts on SWE-bench Verified with Qwen2.5-Coder-7B-Instruct, and how its solved set compares with those of the other routers. Figures 5 and 6 cover all 500 routed episodes.

Figure 5 tracks which expert the router selects at each turn. Every episode opens with `localization`. At turns 5 and 6 the router sends most episodes to `verification`, and from turn 10 onward `localization` again takes 55 to 80% of the turns. The three example episodes show that this average hides frequent switching within a single episode.

Figure 6 measures that switching. The most common run of consecutive turns on one expert lasts a single turn, the median episode switches experts 18 times, and 73% of episodes use two or three of the four adapters.

Figure 7 compares the instances that each routing policy resolves. Foresight shares the most solved instances with GPT-5.5, whose labels it was trained on. The random router resolves the fewest instances and overlaps little with any other policy.



Figure 5: Expert selection by the Foresight router with Qwen2.5-Coder-7B-Instruct on SWE-bench Verified. *(a)* Share of routing decisions per turn by expert. The dashed line counts episodes still running at each turn. *(b)* The expert selected at each turn in three example episodes.

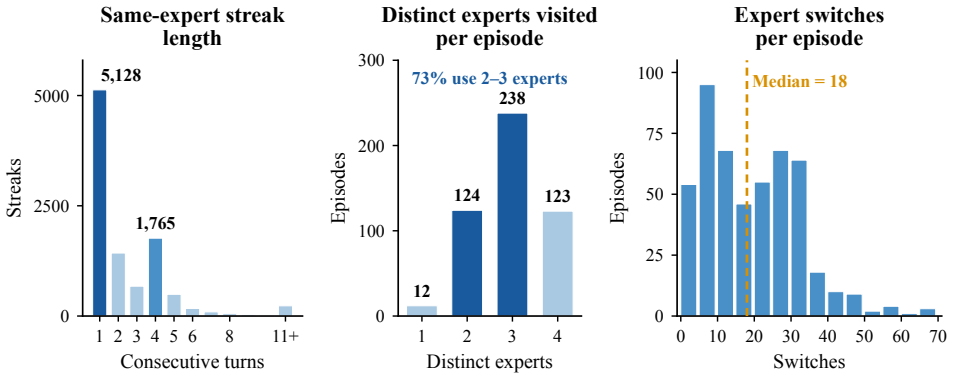


Figure 6: Expert switching by the Foresight router with Qwen2.5-Coder-7B-Instruct on SWE-bench Verified. Left: length of runs of consecutive turns on the same expert. Middle: number of distinct adapters used per episode, counting `fallback`. Right: expert switches per episode.

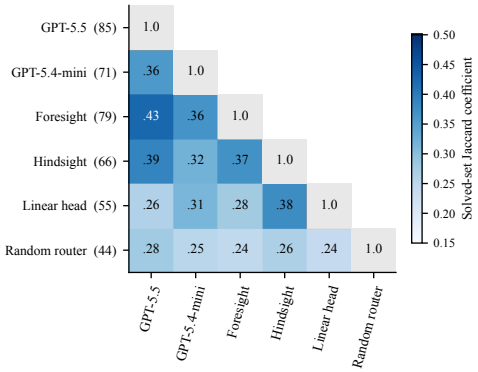


Figure 7: Jaccard overlap between the sets of instances resolved by six routing policies with Qwen2.5-Coder-7B-Instruct on SWE-bench Verified. Numbers in parentheses are resolved counts from Table 2.