LIGHTWEIGHT TOOL PROBING: SHOWING LLM AGENTS WHAT THEIR TOOLS RETURN

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool documentation tells an agent how to call a tool, but only calling it shows what it returns. Multi-turn function calling runs on exactly this missing knowledge. Arguments must match the formats a tool expects, and identifiers returned by one call must feed the next. Existing remedies either keep the agent reading, rewriting the documentation into better text about the tool, or keep it paying, rediscovering tool behavior inside every task; and task success alone cannot show whether such guidance helps. We introduce Lightweight Tool Probing (LTP): before any task, a model probes a toolset once in an isolated sandbox and keeps the raw responses it observed in a frozen evidence bank; during a task, it retrieves at most three short response excerpts per step, while identifiers still come from the live environment. LTP needs no gold solutions, no task feedback, and no parameter updates. To locate where the evidence helps, we also introduce execution diagnostics that score how much of a task's requirements an execution covers. On BFCL, LTP lifts DeepSeek-V4-Flash from 41.75% to 49.75% at 1.09 times the tokens of plain tool calling, outperforming all five baselines, the strongest of which need at least twice the tokens. The gain travels to GLM-5.3 and Qwen3.5-397B, and to ACEBench, AppWorld, and WorkBench. It is interpretable. The tasks LTP rescues gain argument-constraint coverage. Even its trajectories are better training data. They beat plain trajectories for all four student models, by up to 10.75 points. **Showing an agent what its tools actually return is a cheap complement to telling it how to call them.**

1 INTRODUCTION

Language agents increasingly solve tasks by calling tools over many turns. Booking a trip, managing files, or operating a workplace application means chaining calls whose arguments depend on earlier results (Patil et al., 2025; Yao et al., 2024; Trivedi et al., 2024). Tool documentation specifies how to call a tool but rarely shows what it returns, yet the return is exactly what the next call depends on, such as the formats to match, the units to convert, and the identifiers to carry from one call's output into the next call's arguments. Suppose an agent must read a file through an API it has never used (Figure 1). The documentation lists `open(path)` and `read(handle)` but not what a handle looks like, so the agent calls `read` on the file name, fails, retries, and can repeat the mistake on every new task (Figure 1a). Given the same documentation, a model can instead choose what to test, probe it in a sandbox before any task, and record what it observes: that `read` consumes the handle returned by `open`. Every later task can then retrieve that observation (Figure 1b).

Existing remedies fail to let the agent observe a tool once and reuse what it saw without calling the tool again. Documentation rewriting refines tool descriptions (Yuan et al., 2025; Qu et al., 2025a; Fang et al., 2025), but agents must still translate this guidance into valid arguments and bind identifiers across calls. Errors in these steps can cause task failure even when the documentation is accurate. In-task feedback obtains real responses but pays for them inside every task and keeps nothing for the next one (Madaan et al., 2023; Shinn et al., 2023), and RSIAgent builds a memory by exploration but verifies calls on every task (Zhu et al., 2026); in our experiments these remedies cost 1.7 to 2.3 times the tokens of plain tool calling. Training on interaction trajectories (Zeng et al., 2024; Zelikman et al., 2022) moves the knowledge into the weights but needs gold solutions or a success signal. What is missing is a way to expose agents to actual tool outputs rather than rewritten descriptions, with preparation costs incurred once per toolset rather than per task, and independent of

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.

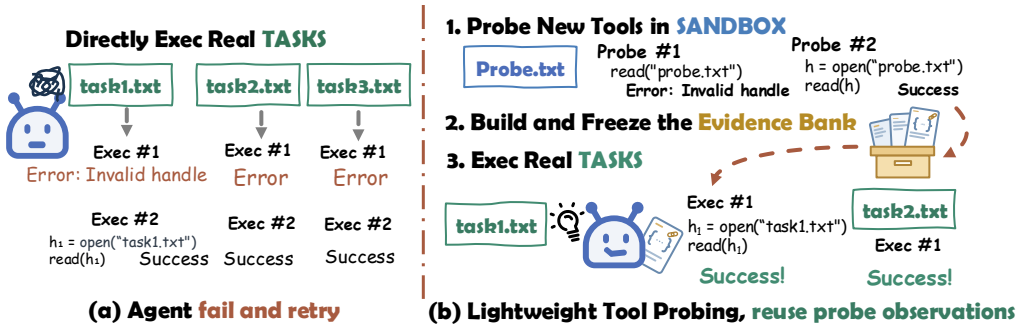


Figure 1: **One sandbox probe replaces repeated in-task failures.** (a) Executing tasks directly, the agent calls `read` on an unopened file, fails, and retries the mistake in every new task. (b) With LTP, the same checkpoint probes the tools once, banks the observation, and succeeds on the first attempt after obtaining its own fresh handle. Illustrative file API: the call sequence is explanatory, not measured, and errors are never delivered as evidence.

labels and training. Studying and improving how agents acquire and use this knowledge raises three challenges. **Challenge 1 (locating the failure).** Task success alone cannot tell whether an agent never investigated a tool, investigated it without obtaining usable evidence, or had the evidence and failed to apply it, and agents often discover useful information yet ignore it (Engländer et al., 2026). **Challenge 2 (transferring evidence across states at low cost).** Tool behavior becomes visible only when a tool is called, yet a response observed in one environment state does not guarantee that the same call succeeds in another task's state. Gathering that evidence must not repeat the work in every task. **Challenge 3 (isolating training transfer).** We must test whether probe-guided trajectories improve SFT outcomes when training tasks are held fixed and no evidence is retrieved at test time.

To address these challenges, we propose Lightweight Tool Probing (LTP). The model finds out how a toolset behaves by probing it once in an isolated sandbox before any task, and keeps the raw responses it observed in a frozen evidence bank, each linked to the call that produced it. During a task, LTP retrieves at most three response excerpts relevant to the current step, within 1,800 characters and marked as local to the sandbox, while the agent still obtains its own identifiers from the live environment (for Challenge 2). It needs no gold solutions, no task success signal, and no parameter updates. Probing pays for itself. On BFCL (Patil et al., 2025) with DeepSeek-V4-Flash, **LTP gains 8.0 points at 1.09 times the tokens of plain tool calling**, beating all five baselines while adding the fewest tokens (Figure 5). The gain is not tied to one model or benchmark. LTP also improves performance with GLM-5.3 and Qwen3.5-397B, as well as on ACEBench, AppWorld, and WorkBench. Execution diagnostics show that argument-constraint coverage increases on tasks LTP rescues and decreases on those it breaks (Section 5.2). The observations even outlive inference. Using LTP-generated trajectories for supervised fine-tuning (SFT) improves performance across all four student models, including a double-digit gain for Qwen3-14B.

Our contributions can be summarized as follows:

- **Diagnostics that say where evidence helps, not only whether a task passed.** Three conditional gaps name the ways a tool goes unused, from never investigating it, to investigating without usable evidence, to having evidence but not applying it. Three coverage scores grade each executed trajectory against the task's requirements; they track success within each of six models and move with the outcomes LTP changes (Sections 3, 5.2; Challenge 1).
- **A cheap way to show an agent what its tools return.** Lightweight Tool Probing (LTP) lets a model try its tools once in a sandbox, before any task, and keep the raw responses it observed. Every later task reuses those observations at a fixed cost per step, with no labels, no task feedback, and no training. On BFCL, it beats five baselines at the lowest overhead (Sections 4, 5.3; Challenge 2).
- **Evidence that also transfers through training, tested fairly.** Fine-tuned on LTP trajectories from the same teacher and tasks, all four student models beat their plain-trajectory controls with no bank at test time. For Qwen3-8B, LTP trajectories also beat trajectories generated by the ToolGrad and ToolACE-MT pipelines from the same teacher (Section 5.4; Challenge 3).

2 RELATED WORK

Learning and evaluating tool use. Tool calling is learned from generated or curated call data (Schick et al., 2023; Qin et al., 2024b; Patil et al., 2024; Tang et al., 2023; Liu et al., 2024b; 2025), and agents are fine-tuned on interaction trajectories (Zeng et al., 2024; Chen et al., 2023; 2024b; Xi et al., 2024; Zhang et al., 2025; Wang et al., 2024a), synthetic instructions (Wang et al., 2023b), correctness-filtered self-generated solutions (Zelikman et al., 2022; Singh et al., 2024), success/failure trajectory preferences (Song et al., 2024), or experience from interaction and search (Su et al., 2025; Yu et al., 2025; Li et al., 2026; Zhao et al., 2026); surveys map this space (Qin et al., 2024a; Qu et al., 2025b). Our SFT study compares jointly successful trajectories with and without probe evidence. Benchmarks cover call correctness (Li et al., 2023; Zhuang et al., 2023; Huang et al., 2024b), step-level utilization (Chen et al., 2024a), stable API evaluation (Guo et al., 2024), stateful multi-turn calling (Lu et al., 2025; Patil et al., 2025; Yao et al., 2024), and interactive environments (Liu et al., 2024a; Zhou et al., 2024; Farn & Shin, 2023; Trivedi et al., 2024). Most score outcomes; our diagnostics check whether an execution obtained, constrained, and bound the evidence its requirements demand, a verifier-free analogue of process supervision (Lightman et al., 2024).

Tool documentation and memory. Documentation alone enables zero-shot tool use (Hsieh et al., 2023); EasyTool condenses it into instructions (Yuan et al., 2025), and DRAFT and Play2Prompt refine it through tool interaction (Qu et al., 2025a; Fang et al., 2025). Other work distills reusable experience into skills and insights (Wang et al., 2023a; Zhao et al., 2024), workflows (Wang et al., 2024b), episodic memory (Park et al., 2023), or tool-side memory of capabilities and failure boundaries (Fang et al., 2026). Both lines hand the agent rewritten descriptions of tool behavior. Like DRAFT, Play2Prompt, and ToolAtlas, LTP probes reusably, but hands it the outputs themselves: raw, trace-linked response excerpts from the model's own bank, retrieved under a fixed budget (Lewis et al., 2020).

Feedback and exploration at inference. Self-Refine, Reflexion, CRITIC, and tree search revise or expand ReAct-style trajectories (Yao et al., 2023b) with in-task feedback (Madaan et al., 2023; Shinn et al., 2023; Gou et al., 2024; Yao et al., 2023a; Zhou et al., 2023), though unaided self-correction has limits (Huang et al., 2024a). Simulators vet risky actions or refine calls before execution (Ruan et al., 2024; Zhang et al., 2026), and RSIAgent explores a new environment to build memory without training (Zhu et al., 2026), but still verifies calls inside every task. Curiosity-driven and archive-based exploration (Pathak et al., 2017; Ecoffet et al., 2021) and the finding that agents discover but ignore injected solutions (Engländer et al., 2026) motivate separating investigation, evidence, and application. LTP instead pays once, before the first task, from public probe responses, and adds no revision, simulation, or verifier calls.

3 MEASURING THE EXPLORATION-APPLICATION GAP

3.1 THREE CONDITIONAL GAPS

Challenge 1 is to tell three failures apart: the agent never investigated a tool, investigated it without obtaining usable evidence, or had the evidence and failed to apply it. We make the three cases measurable through observable event chains rather than latent knowledge. Each chain tracks a tool condition independently specified in the annotation protocol. Before any task, D records *exposure* to the tool's interface or documentation. I records a relevant *investigation*, and Q records *qualified public probe evidence* supporting that condition. In the file example of Figure 1, seeing `read` in the documentation is exposure, calling it in the sandbox is an investigation, and the recorded observation that `read` consumes the handle returned by `open` is qualified evidence. At task time, O denotes an applicable execution *opportunity*. T means *delivery* of the evidence before the relevant action, and U represents *correct use* under the registered condition. The three gaps are the conditional failure rates along this chain:

$$G_1 = 1 - \Pr(I = 1 \mid D = 1), \tag{1}$$

$$G_2 = 1 - \Pr(Q = 1 \mid I = 1), \tag{2}$$

$$G_3 = 1 - \Pr(U = 1 \mid Q = 1, O = 1). \tag{3}$$

In words, G_1 is how often exposure prompts no investigation. G_2 is how often investigation yields no usable evidence, and G_3 is how often evidence with an applicable opportunity is still not used correctly. The unit of analysis for G_1 and G_2 is a condition paired with a probing checkpoint; for G_3 it is a task-execution opportunity for that condition. Evidence can be available without being delivered, and an agent can act correctly without receiving it, so G_3 alone does not identify the causal effect of retrieved evidence. These gaps frame the execution diagnostics defined below, which Section 5.2 evaluates; the supplementary material specifies the annotation protocol.

3.2 Execution diagnostics

To locate where evidence helps inside an execution, we score how much of a task's requirements it covers, at three levels of strictness. The requirements are annotated once, before any experiment: each task's mandatory conditions, with their valid call paths and argument constraints. $\mathcal{G}_\tau$ collects the requirements that have already appeared in the user dialogue of trajectory τ, so a requirement counts only after the user has asked for it, and $N = |\mathcal{G}_\tau|$. Each requirement g then passes or fails three nested tests, each stricter than the last. *Response* ($r_g = 1$) means some annotated valid call path for g received a qualified (nonempty, non-error) response. *Argument* ($p_g = 1$) means that call also satisfies its directly checkable argument constraints. *Binding* ($b_g = 1$) means every annotated argument is additionally supported by the trace, as a fixed value or a reference to an earlier output, such as a handle returned by `open`. Averaging over $\mathcal{G}_\tau$ gives response, argument-constraint, and binding coverage:

$$
\begin{aligned}
R(\tau) &= \tfrac{1}{N} \textstyle\sum_{g \in \mathcal{G}_\tau} r_g, \\
P(\tau) &= \tfrac{1}{N} \textstyle\sum_{g \in \mathcal{G}_\tau} p_g, \\
B(\tau) &= \tfrac{1}{N} \textstyle\sum_{g \in \mathcal{G}_\tau} b_g.
\end{aligned}
\tag{4}
$$

Each requirement counts at most once within its valid time window, so $B \le P \le R$, and the scores are defined only for valid executions with $N > 0$. Because they credit partial progress, they complement accuracy rather than replace it (Figure 2b).

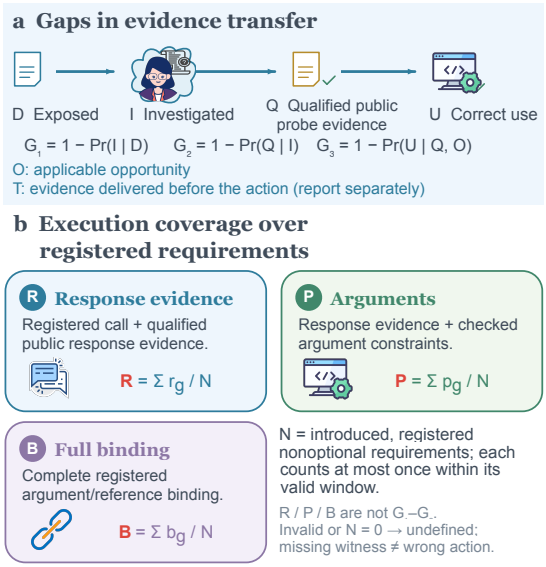


Figure 2: **Where tool use fails, and how we score an execution.** (a) Three conditional gaps: after exposure (D), an agent may never investigate a tool (I), investigate without obtaining usable evidence (Q), or obtain evidence and still not use it correctly (U). (b) Three nested coverage scores of an executed trajectory, $B \le P \le R$: responses obtained, argument constraints met, and references bound. The scores grade executions and do not estimate the gaps; both panels are schematic.

4 Lightweight Tool Probing

LTP is a fixed protocol for prompting, evidence construction, and retrieval. It trains no probing policy and updates no parameters. It assumes public tool descriptions and schemas, observable tool responses, and an isolated, resettable sandbox. For a model θ, task x, visible interaction history h_t, and sandbox E_s, LTP computes

$$
\mathcal{B}_\theta = \mathrm{Build}\left(\mathrm{Probe}_\theta(E_s)\right), \tag{5}
$$

$$
e_t = \mathrm{Retrieve}\left(\mathcal{B}_\theta, h_t, \text{visible schemas}\right), \qquad a_t \sim p_\theta(\cdot \mid x, h_t, e_t). \tag{6}
$$

The bank $\mathcal{B}_\theta$ is built once, before any task; the evidence e_t is recomputed at each decision t and conditions the next action a_t. Probing and execution share one fixed θ, and the intervention is purely in-context: Plain is the special case $e_t \equiv \varnothing$. Figure 3(a) shows the four stages.

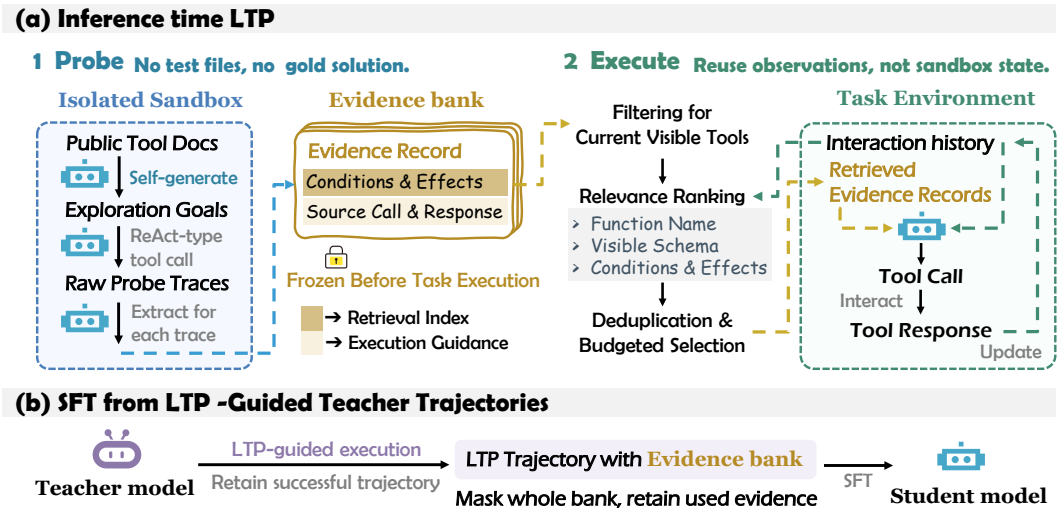


Figure 3: **LTP probes once and retrieves raw responses.** (a) Inference: the same fixed checkpoint (1) explores the public tools once in an isolated sandbox, (2) stores each observed response with its source call in a frozen evidence bank, (3) retrieves at most three eligible records within 1,800 characters at each decision, and (4) acts in the task environment, obtaining its own state-dependent identifiers there. (b) Paired SFT: the same teacher and Train400 tasks yield 127 jointly successful Plain/LTP trajectory pairs; students train on either set, with the retrieved evidence kept as loss-masked context in LTP trajectories, and every checkpoint is evaluated with Plain inference and with its own bank.

(1) Probing once. The model proposes exploration goals from the public documentation, covering prerequisites, state changes, errors, and recovery, and pursues them in a fixed number of probe episodes (80 on BFCL) under fixed interaction budgets. Each episode starts from a fresh, task-independent sandbox state, and probing sees only public tool responses, never gold solutions, evaluator state, or benchmark test files.

(2) A bank of raw observations. Each record of $\mathcal{B}_\theta$ is a tuple $\rho = (f, c, r, \delta)$: a tool f, a call c issued during probing, the raw response r it returned, and a short description δ of the observed conditions and effects, written by the same model. δ serves only as a retrieval index; what LTP delivers is r itself: the details that decide an argument, field names, value formats, identifier shapes, are exactly what a summary can drop (for Challenge 2). The extraction prompt keeps negative observations and uncertainty. The bank is frozen before any task and shared by all tasks.

(3) Budgeted retrieval. At each decision t, LTP scores the records of the currently visible tools. A record is *eligible* if r is a nonempty, non-error JSON object or array and the arguments of c pass the visible schema checks; error records stay in the bank but are never delivered, so that no excerpt demonstrates a failed call. Eligible records are ranked by lexical overlap with the user and tool messages in h_t, comparing the name of f, then its schema, then δ, and keeping one record per tool. e_t is then the longest ranked prefix whose rendered text, with a caution that all values are local to the sandbox, fits k records and m characters; if nothing fits, $e_t = \varnothing$ and the step is exactly Plain. We use $k = 3$ and $m = 1{,}800$ throughout.

(4) Execution with the task's own state. The agent then acts through the benchmark's tool-calling interface. A delivered r transfers what is environment-invariant, the field names, formats, and prerequisite structure a response exposes; state-bound values, such as handles or booking numbers, are re-derived from the agent's own calls in the live task (for Challenge 2).

Cost and properties. Probing and bank construction run once per model and environment, so their cost is shared by all tasks; at task time, LTP adds at most m characters per decision and no extra model or tool calls. On BFCL, preparation takes 8.18 million tokens, and LTPof the whole protocol uses 1.09 times the tokens of Plain (Section 5.3; for Challenge 2). Because the bank is frozen before the first task and retrieval is deterministic, the evidence never adapts to a task, and $e_t = \varnothing$ recovers Plain when the bank holds nothing relevant.

5 EXPERIMENTS

Three research questions follow the three challenges. RQ1 asks whether the execution diagnostics separate successful from failed executions and move when retrieval changes an outcome (Section 5.2). RQ2 asks whether LTP improves accuracy at low token cost and carries over to other models and environments (Section 5.3). RQ3 asks with tasks and teacher held fixed, whether LTP trajectories make better SFT data (Section 5.4).

5.1 SETUP

Benchmark and metrics. We use the four multi-turn categories of BFCL (Patil et al., 2025) (base, missing function, missing parameter, and long context) in two 400-task panels, Train400 and Test400. Inference is evaluated on all 800 tasks; SFT trains on Train400 trajectories and is evaluated on Test400. Out-of-distribution evaluations use ACEBench Agent (Chen et al., 2025), AppWorld Test (Trivedi et al., 2024), and WorkBench (Styles et al., 2024). Accuracy is the official task success over every task. The diagnostics R, P, and B use the requirement annotations of Section 3.2, held fixed across the reported analyses, and are defined only for valid executions, those the scorer can parse with complete logs. We summarize their association with success by the area under the ROC curve (AUROC), computed from valid Plain executions over all 800 tasks. Paired diagnostic changes use tasks valid under both Plain and LTP.

Models and conditions. DeepSeek-V4-Flash (the 0731 version) is the main inference model and the SFT teacher. Students are Qwen2.5-7B, Qwen3-8B, Qwen3.5-9B, and Qwen3-14B (Yang et al., 2025). Qwen3.5-397B-A17B and GLM-5.3(Team GLM et al., 2024) are used in the cross-model and diagnostic panels. *Plain* calls the benchmark tools directly; *LTP* adds the checkpoint's own bank from 80 probe episodes with $k = 3$ and $m = 1{,}800$ (Section 4). Baselines are BFCL adaptations of EasyTool, Play2Prompt, Reflexion, RSIAgent (Zhu et al., 2026), and Gecko/GATS (Zhang et al., 2026). Token cost counts the evaluated execution, its helper calls, and preparation amortized over the tasks that share it, reported as a ratio to Plain.

Training. We keep the 127 teacher–task pairs from Train400 that succeed under both Plain and LTP generation (Figure 3b), so that both training arms share the same tasks and teacher (for Challenge 3). *Plain SFT* uses the Plain trajectories; *Retained SFT* uses the paired LTP trajectories and keeps the bank context as loss-masked input. All runs use global batch size 8 and an assistant-only loss. Table 4 reports Qwen2.5-7B at learning rate 5×10^{-6}, epoch 1; Qwen3-8B at 2×10^{-6}, epoch 3; Qwen3.5-9B at 2×10^{-6}, epoch 2; and Qwen3-14B at 2×10^{-6}, epoch 6, with Plain and Retained SFT matched on learning rate, epoch, and the other training settings (Appendix A). Every checkpoint is evaluated with Plain inference and with LTP using its own bank. Figure 6 adds two Qwen3-8B students trained on trajectories that the same teacher generates for the Train400 tasks through fixed-task adaptations of ToolGrad (Zhou et al., 2026), which builds tool-call chains by textual-gradient refinement, and ToolACE-MT (Zeng et al., 2025), which drafts a whole multi-turn dialogue and refines it by mask-and-fill. Both generation pipelines produce 127 trajectories that pass the official task checker. One ToolGrad trajectory has a serialized training sequence of 167,266 tokens, exceeding the configured 163,840-token limit. We exclude this entire trajectory without truncation, leaving 126 ToolGrad trajectories and 127 ToolACE-MT trajectories for SFT.

5.2 DO THE DIAGNOSTICS CAPTURE EXECUTION QUALITY? (RQ1)

Coverage separates successful from failed executions within every model and moves with outcome changes. Under Plain inference, argument-constraint coverage P separates successful from failed executions with AUROC 0.84 to 0.86 for Qwen2.5-7B, Qwen3-8B, and Qwen3-14B and 0.65 to 0.75 for Qwen3.5-9B, Qwen3.5-397B, and DeepSeek-V4-Flash; R shows the same separation between these groups, and B ranges from 0.64 to 0.76 (Table 1). AUROC ranks a model's own executions rather than comparing models, and it is lower for the stronger models, likely because they already satisfy most constraints (mean P of 0.76 for Qwen3.5-9B and 0.79 for DeepSeek-V4-Flash, versus 0.64 for Qwen2.5-7B and 0.59 for Qwen3-8B), which leaves less variation between their successes and failures. The diagnostics also move with the outcome of each task. On tasks that LTP rescues, mean ΔR, ΔP, and ΔB are positive for every model, and on tasks it breaks they are negative (Figure 4).

Table 1: **Coverage separates successful from failed executions within every model.** Plain-inference AUROC on BFCL (Train400 + Test400) for predicting task success from response (R), argument-constraint (P), and binding (B) coverage; 0.5 is chance. Six models, one run each; sample sizes in Appendix A, two further models in Appendix C.

Score	Qwen2.5-7B	Qwen3-8B	Qwen3.5-9B	Qwen3-14B	Qwen3.5-397B	DeepSeek-V4-Flash
R (response)	0.827	0.846	0.745	0.846	0.711	0.590
P (argument)	0.849	0.859	0.747	0.846	0.720	0.655
B (binding)	0.711	0.736	0.686	0.763	0.684	0.642

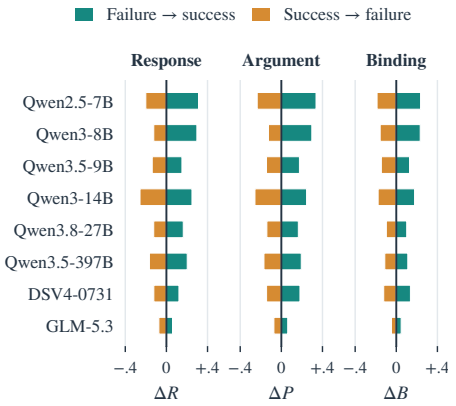


Figure 4: **When LTP flips an outcome, coverage moves with it.** Mean within-task change in R, P, and B (LTP minus Plain) on tasks that flip from failure to success and from success to failure, for eight models on BFCL.

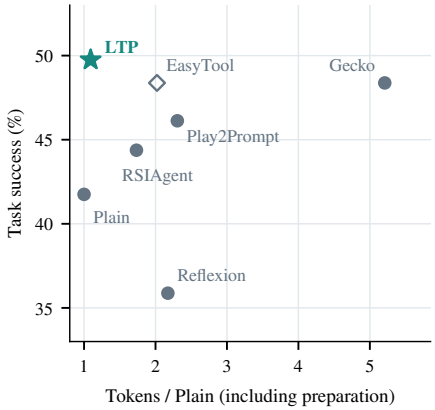


Figure 5: **LTP is the most accurate method and adds the fewest tokens.** Task success against tokens relative to Plain for the seven methods of Table 2; EasyTool is drawn as an open diamond.

5.3 Does LTP improve inference accuracy, and at what cost? (RQ2)

LTP gains 8.0 points with 9.2% more tokens than Plain. LTP raises the solved tasks from Plain's 334 (41.75%) to 398 (49.75%), gaining 3.5 to 12.5 points in every category (Table 2). This costs 118.02 million tokens in total, 109.85 million for execution and 8.18 million for preparation, or 1.092 times Plain's 108.05 million. Every baseline trails LTP in accuracy while spending more: EasyTool and Gecko/GATS come closest at 48.38% (Gecko/GATS: 387 tasks) but need 2.021 and 5.209 times Plain's tokens (Gecko/GATS: 562.80 million); Play2Prompt reaches 46.13% at 2.305 times, RSIAgent 44.38% at 1.734 times, and Reflexion 35.88% at 2.173 times.

Table 2 reports results on all BFCL tasks. Diagnostic means use the 795 valid tasks with defined scores for all seven methods. LTP reaches $(R, P, B) = (.848, .797, .598)$, compared with Plain's $(.843, .788, .594)$. LTP thus raises task accuracy and mean coverage over Plain at once, although RSIAgent and EasyTool post higher means on some diagnostics. At the task level, LTP rescues 121 of the 797 tasks with valid diagnostics under both conditions and breaks 58; the rescued tasks gain .165 in P on average, and the broken tasks lose .225. Figure 4 reports fixed Plain/LTP comparisons over the same 800-task suite.

On the full BFCL evaluation, LTP improves Qwen3.5-397B-A17B from 441 to 454 successful tasks (+1.63 percentage points) and GLM-5.3 from 433 to 469 (+4.50 points; Table 3). Across environments, DeepSeek-V4-Flash-0731 gains 6.12 points on ACEBench Agent (Chen et al., 2025), 7.52 on AppWorld Test (Trivedi et al., 2024), and 5.24 on WorkBench (Styles et al., 2024). Appendix B.1 illustrates how diagnostic changes correspond to recorded tool calls, and Appendix B.2 presents three tasks on which LTP succeeds while the named alternatives fail.

Table 2: **LTP reaches the highest accuracy at the smallest token overhead.** DeepSeek-V4-Flash on the 800 BFCL multi-turn tasks (200 per category). Acc.: task success (%); Tok.: execution, helper, and amortized preparation tokens relative to Plain; $R/P/B$: response, argument-constraint, and binding coverage.

	Base		Missing func.		Missing param.		Long context		Overall				
Method	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	R	P	B
Plain	54.00	1.000	31.50	1.000	32.00	1.000	49.50	1.000	41.75	1.000	0.843	0.788	0.594
Reflexion	44.00	2.203	34.00	2.139	30.50	2.138	35.00	2.203	35.88	2.173	0.797	0.744	0.567
RSIAgent	57.50	1.593	39.50	1.658	29.50	1.775	51.00	1.823	44.38	1.734	**0.855**	**0.798**	**0.601**
Play2Prompt	53.00	3.182	33.50	2.321	**46.00**	2.783	52.00	1.644	46.13	2.305	0.795	0.770	0.566
EasyTool	58.50	2.389	**43.50**	2.204	42.00	2.279	49.50	1.590	48.38	2.021	0.832	**0.798**	0.600
Gecko/GATS	54.50	6.533	39.00	6.376	41.00	6.763	**59.00**	2.952	48.38	5.209	0.835	0.789	0.591
LTP (ours)	**61.50**	**1.118**	40.00	**1.097**	44.50	**1.091**	53.00	**1.079**	**49.75**	**1.092**	0.848	0.797	0.598

Bold/underline: best/second value per column, Plain excluded from the token ranking. $R/P/B$ use the 795 tasks valid for all methods; accuracy uses all 800.

Table 3: **The gain carries over to other models and other environments.** (a) BFCL, all 800 tasks, for three models. (b) Three further environments with DeepSeek-V4-Flash, scored by each environment's own checker. Values are task success (%); Δ is LTP minus Plain.

(a) BFCL 800: across models

Model	Plain	LTP	Δ
DeepSeek-V4-Flash	41.75	49.75	+8.00
Qwen3.5-397B	55.13	56.75	+1.63
GLM-5.3	54.13	58.63	+4.50

(b) OOD: across environments

Environment	N	Plain	LTP	Δ
ACE Agent	49	24.49	30.61	+6.12
AppWorld Test	585	13.33	20.85	+7.52
WorkBench	210	74.29	79.52	+5.24

(a) GLM-5.3 runs with reasoning enabled at low effort. (b) ACE: 49 English Agent tasks (19 multi-step + 30 multi-turn); AppWorld: all 585 official Test tasks; WorkBench: all 210 official multi-domain tasks.

5.4 Do LTP trajectories make better SFT data? (RQ3)

LTP trajectories can make better SFT data. Under Plain inference, the Retained SFT checkpoints beat their matched Plain SFT controls for all four students, by 4.25 to 10.75 points; the largest gap is Qwen3-14B, with 42.00% versus 31.25% (Table 4). The checkpoints also satisfy more argument constraints and bind more references than their controls. P and B rise for all four students and R for three, with a 0.003 drop for Qwen3.5-9B (Figure 7). LTP-generated trajectories can therefore improve task performance even when the student receives no bank at inference. At the same checkpoints, adding each student's own bank lowers accuracy by 3.00–5.00 points for three students and raises it by 0.50 points for Qwen3-14B (Table 4).

The smaller gains for Qwen3.5-9B may reflect a smaller remaining gap in tool-use knowledge. Its Plain SFT baseline already achieves 48.50% accuracy and .835 response coverage, leaving less additional benefit from LTP-generated trajectories than for the other students. The small decrease in R accompanies improvements in argument constraints, bindings, and task success.

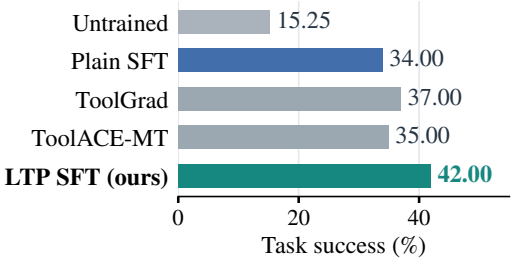
More broadly, the benefit of inference-time LTP depends on what the model already knows about the tools. SFT on LTP trajectories may internalize part of the knowledge that retrieval otherwise supplies, reducing the additional information provided by the bank. When this benefit becomes small, interference from the added context may outweigh it. For example, on `multi_turn_miss_func_23`, the fine-tuned Qwen3.5-9B completes the task under Plain inference, but repeatedly reads an empty file under LTP and fails to advance to the subsequent steps. At this checkpoint, LTP rescues 22 tasks but turns 34 successes into failures, yielding a net accuracy decrease of 3.00 percentage points.

For Qwen3-8B, LTP trajectories also beat trajectories generated by other pipelines from the same teacher. Under Plain inference on Test400 (Figure 6), Retained SFT reaches 42.00%, against 34.00% for Plain SFT, 37.00% for ToolGrad, and 35.00% for ToolACE-MT; the untrained model

Table 4: **Training on LTP trajectories beats training on plain trajectories for every student.** Matched SFT on 127 paired Train400 trajectories, evaluated on Test400 (% task success). P/L: Plain inference / LTP inference with the student's own bank; Ours: Retained SFT; Δ: Plain-inference gain of ours over Plain SFT.

Student	Untrained P/L	Plain SFT P/L	Ours P/L	Δ
Qwen2.5-7B	11.50/14.00	22.75/22.50	30.00/25.75	+7.25
Qwen3-8B	15.25/18.50	34.00/35.75	42.00/37.00	+8.00
Qwen3.5-9B	28.75/29.50	48.50/46.75	52.75/49.75	+4.25
Qwen3-14B	13.25/19.00	31.25/30.25	42.00/42.50	+10.75

Figure 6: **LTP trajectories beat trajectories from other generation pipelines.** Qwen3-8B students under Plain inference on Test400, task success (%); Plain SFT, Retained SFT, and the untrained model are the Table 4 entries.



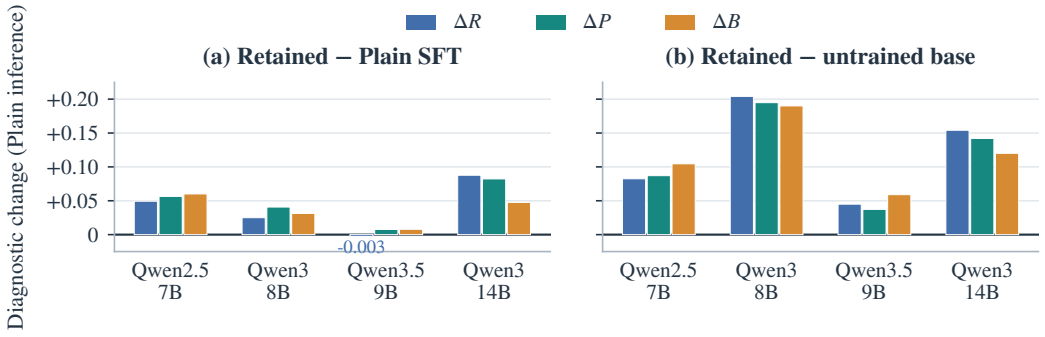


Figure 7: **Retained SFT raises argument-constraint and binding coverage for every student.** Diagnostic changes of the Retained SFT checkpoints of Table 4 under Plain inference, relative to (a) Plain SFT and (b) the untrained model, on each pair's jointly valid tasks.

scores 15.25%. The advantage occurs with the same teacher and comparable trajectory counts, so neither more data nor a stronger teacher explains it. Appendix B.3 shows three cases from the matched Plain versus Retained SFT comparison under Plain inference.

6 Conclusion

LTP starts from a simple asymmetry: documentation tells an agent how to call a tool, but only calling it shows what it returns. The protocol therefore probes a toolset once in an isolated sandbox, banks the raw responses rather than rewritten summaries, and retrieves at most a few short excerpts per decision, while identifiers and other task state still come from the live environment. The cost of learning how a toolset behaves is thus paid once per toolset, never inside a task, and the intervention stays purely in-context: no gold solutions, no task feedback, no parameter updates. On BFCL, LTP raises DeepSeek-V4-Flash by 8.0 points at 1.09 times the tokens of plain tool calling, while the five baselines spend 1.7 to 5.2 times the tokens for less, and the gain carries over to two other models, and to three other environments. The execution diagnostics show that the gain is no accident of the checker: on average, the tasks LTP rescues satisfy more argument constraints and the tasks it breaks satisfy fewer. The observations even outlive inference: with tasks and teacher matched, checkpoints fine-tuned on LTP trajectories beat their matched controls for all four student models, without any bank at test time. Together, these results argue for a division of labor: sandbox evidence carries what is invariant about a tool, the live task supplies what is not, and either a retrieval step or the training data can deliver the evidence. **Showing an agent what its tools actually return is a cheap complement to telling it how to call them.**

AI USE STATEMENT

We used large language models (LLMs) as assistants for experiment planning, code implementation, trajectory processing, result analysis, literature search, manuscript drafting, and figure generation. Separately from this assistance, LLMs are part of the experiments themselves: they are the evaluated agents, they generate the teacher trajectories used for fine-tuning, and they produced the initial requirement annotations behind the execution diagnostics. The authors reviewed all AI-assisted text, code, and figures, checked the reported results against the experiment records, and take full responsibility for the content of this paper.

REPRODUCIBILITY STATEMENT

Sections 3 and 4 define the diagnostics and LTP; Section 5.1 states the models, task panels, baselines, and training settings; Appendix A adds benchmark handling, valid sample counts, cost accounting, and seeds. All benchmarks and models are publicly available. The supplementary material contains our code for probing, bank construction, retrieval, the diagnostics, the baseline adaptations, and SFT.

REFERENCES

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Yuefeng Huang, Xiangcheng Liu, Xinzhi Wang, et al. Acebench: A comprehensive evaluation of llm tool usage. In *EMNLP (Findings)*, pp. 12970–12998, 2025.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, et al. T-eval: Evaluating the tool utilization capability of large language models step by step. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9510–9529, 2024a.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. Agent-FLAN: Designing data and methods of effective agent tuning for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024b. URL https://arxiv.org/abs/2403.12881.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. First return, then explore. *Nature*, 590:580–586, 2021. URL https://arxiv.org/abs/2004.12919.

Leon Engländer, Sophia Althammer, Ahmet Üstün, Matthias Gallé, and Tom Sherborne. Agents explore but agents ignore: Llms lack environmental curiosity. *arXiv preprint arXiv:2604.17609*, 2026.

Wei Fang, Yang Zhang, Kaizhi Qian, James Glass, and Yada Zhu. Play2prompt: Zero-shot tool instruction optimization for llm agents via tool play. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 26274–26290, 2025.

Yue Fang, Zhibang Yang, Fangkai Yang, Xiaoting Qin, Liqun Li, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Toolatlas: Learning once, reusing everywhere with tool-side memory. *arXiv preprint arXiv:2607.11126*, 2026.

Nicholas Farn and Richard Shin. Tooltalk: Evaluating tool-usage in a conversational setting. *arXiv preprint arXiv:2311.10775*, 2023.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2305.11738.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11143–11156, 2024.

Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. Tool documentation enables zero-shot tool-usage with large language models. *arXiv preprint arXiv:2308.00675*, 2023.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations (ICLR)*, 2024a. URL https://arxiv.org/abs/2310.01798.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Gong, et al. Metatool benchmark for large language models: Deciding whether to use tools and which to use. In *International Conference on Learning Representations*, volume 2024, pp. 42978–43007, 2024b.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL https://arxiv.org/abs/2005.11401.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 3102–3116, 2023.

Zeping Li, Hongru Wang, Yiwen Zhao, Guanhua Chen, Yixia Li, Keyang Chen, Yixin Cao, Guangnan Ye, Hongfeng Chai, and Zhenfei Yin. Rethinking the role of entropy in optimizing tool-use behaviors for large language model agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27955–27967, 2026.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2305.20050.

Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, et al. Toolace: Winning the points of llm function calling. In *International conference on learning representations*, volume 2025, pp. 41359–41381, 2025.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024a.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, et al. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *Advances in Neural Information Processing Systems*, 37:54463–54482, 2024b.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*, 2023. URL https://arxiv.org/abs/2304.03442.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, 2017. URL https://arxiv.org/abs/1705.05363.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565, 2024.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, et al. Tool learning with foundation models. *ACM Computing Surveys*, 57(4):1–40, 2024a.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pp. 9695–9717, 2024b.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. From exploration to mastery: Enabling llms to master tools via self-driven interactions. In *International Conference on Learning Representations*, volume 2025, pp. 55675–55697, 2025a.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343, 2025b.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of LM agents with an LM-emulated sandbox. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2309.15817.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

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

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, Abhishek Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura Culp, Lechao Xiao, Maxwell L. Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024. URL https://arxiv.org/abs/2312.06585.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization for LLM agents. In *Proceedings of ACL 2024 (Volume 1: Long Papers)*, 2024. URL https://arxiv.org/abs/2403.02502.

Olly Styles, Sam Miller, Patricio Cerda-Mardini, Tanaya Guha, Victor Sanchez, and Bertie Vidgen. WorkBench: a benchmark dataset for agents in a realistic workplace setting. In *First Conference on Language Modeling (COLM)*, 2024. URL https://openreview.net/forum?id=4HNAwZFDcH.

Hongjin Su, Ruoxi Sun, Jinsung Yoon, Pengcheng Yin, Tao Yu, and Sercan Arik. Learn-by-interact: A data-centric framework for self-adaptive agents in realistic environments. In *International Conference on Learning Representations*, volume 2025, pp. 89044–89097, 2025.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*, 2023.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Jingyu Sun, Juanzi Li, Lei Zhao, Lindong Wu, Lucen Zhong, Mingdao Liu, Minlie Huang, Peng Zhang, Qinkai Zheng, Rui Lu, Shuaiqi Duan, Shudan Zhang, Shulin Cao, Shuxun Yang, Weng Lam Tam, Wenyi Zhao, Xiao Liu, Xiao Xia, Xiaohan Zhang, Xiaotao Gu, Xin Lv, Xinghan Liu, Xinyi Liu, Xinyue Yang, Xixuan Song, Xunkai Zhang, Yifan An, Yifan Xu, Yilin Niu, Yuantao Yang, Yueyan Li, Yushi Bai, Yuxiao Dong, Zehan Qi, Zhaoyu Wang, Zhen Yang, Zhengxiao Du, Zhenyu Hou, and Zihan Wang. ChatGLM: A family of large language models from GLM-130B to GLM-4 all tools, 2024. URL https://arxiv.org/abs/2406.12793.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023a. URL https://arxiv.org/abs/2305.16291.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*, 2024a.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of ACL 2023 (Volume 1: Long Papers)*, 2023b. URL https://arxiv.org/abs/2212.10560.

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

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. AgentGym: Evolving large language model-based agents across diverse environments, 2024. URL https://arxiv.org/abs/2406.04151.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a. URL https://arxiv.org/abs/2305.10601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023b. URL https://arxiv.org/abs/2210.03629.

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

Xiao Yu, Baolin Peng, Vineeth Vajipey, Hao Cheng, Michel Galley, Jianfeng Gao, and Zhou Yu. Exact: Teaching ai agents to explore with reflective-mcts and exploratory learning. In *International Conference on Learning Representations*, volume 2025, pp. 65157–65184, 2025.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Kan Ren, Dongsheng Li, and Deqing Yang. Easytool: Enhancing llm-based agents with concise tool instruction. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 951–972, 2025.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2203.14465.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077, 2024.

Xingshan Zeng, Weiwen Liu, Lingzhi Wang, Liangyou Li, Fei Mi, Yasheng Wang, Lifeng Shang, Xin Jiang, and Qun Liu. ToolACE-MT: Non-autoregressive generation for agentic multi-turn interaction, 2025. URL https://arxiv.org/abs/2508.12685.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, Zhiwei Liu, Yihao Feng, Tulika Awalgaonkar, Rithesh Murthy, Eric Hu, Zeyuan Chen, Ran Xu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. xLAM: A family of large action models to empower AI agent systems. In *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2025. URL https://arxiv.org/abs/2409.03215.

Zeyu Zhang, Guohao Li, Zhenchang Xing, Alexandros Apostolopoulos, Yu Lin Lee, and Liang Zheng. Gecko: A simulation environment with stateful feedback for refining agent tool calls, 2026. URL https://arxiv.org/abs/2602.19218.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024. URL https://arxiv.org/abs/2308.10144.

Yiwen Zhao, Zhihao Wen, Yuchen Mao, Mingxuan Jiang, Yihao Hu, Pan Wang, Xin Zhang, and Wei Wu. Towards better agents for multi-turn user interaction: The next user turn is more than context. *arXiv preprint arXiv:2608.17499*, 2026.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models, 2023. URL https://arxiv.org/abs/2310.04406.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2307.13854.

Zhongyi Zhou, Kohei Uehara, Haoyu Zhang, Jingtao Zhou, Lin Gu, Ruofei Du, Zheng Xu, and Tatsuya Harada. ToolGrad: Efficient tool-use dataset generation with textual "gradients". In *Findings of ACL 2026*, 2026. URL https://arxiv.org/abs/2508.04086.

Sibo Zhu, Shicheng Fan, Xinyue Wang, Wenyi Wu, Kun Zhou, and Biwei Huang. RSIAgent: Autonomous exploration for recursive self-improvement in new environments, 2026. URL https://arxiv.org/abs/2609.15364.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. ToolQA: A dataset for LLM question answering with external tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2306.13304.

A EXPERIMENTAL DETAILS

Benchmark handling. Accuracy uses the official BFCL scorer on all tasks and counts context exhaustion and residual transport failures as failures. Executions interrupted by transport errors are rerun under fixed rules, and only the final execution enters accuracy and cost.

Diagnostics. R, P, and B use the fixed requirement annotations and scoring rules without refitting. Table 1 and Figure 4 pool the disjoint Train400 and Test400 tasks; AUROC uses tasks with defined Plain diagnostics, paired changes use tasks with defined diagnostics in both executions, and undefined scores are excluded, never set to zero. Figure 4 uses untrained checkpoints for all eight models, averaging within-run transition means over three repeats each for Qwen3.8-27B and GLM-5.3 and using one Plain/LTP pair for each other model. For Table 1, the valid sample sizes (n) out of 800 tasks per model are Qwen2.5-7B: 794; Qwen3-8B: 795; Qwen3.5-9B: 797; Qwen3-14B: 798; Qwen3.5-397B: 798; DeepSeek-V4-Flash: 798. These counts apply to all three diagnostics.

Baselines. All baselines are adapted to the BFCL interface. Reflexion uses public feedback and a fixed second trial, is scored on that trial, and is charged for the first trial and the reflection; RSIAgent is charged for its verifier calls. Gecko/GATS covers all 800 tasks; five tasks with incomplete usage records were rerun, replacing their outcomes, diagnostics, and costs together. Inference seeds in Table 2: Plain 116931, Gecko/GATS 116933, LTP run 116939.

Cost accounting. Token cost counts the input and output tokens of the evaluated execution and its method-specific helper calls, plus preparation counted once and shared by the tasks that use it. Failed tasks stay in the denominator, and their costs count. Gecko's helper tokens are attributed to the trajectories; helper tokens from superseded attempts are excluded.

Training. The loss is token-normalized over assistant decisions, with the preceding history as context. The training seed is 42 for the Qwen2.5-7B, Qwen3-8B, and Qwen3.5-9B students in Table 4 and for the fine-tuned Qwen3-8B checkpoints in Appendix B; the Qwen3-14B student uses seed 5.

B CASE STUDIES FROM RECORDED TRAJECTORIES

B.1 TRACE EXAMPLES FOR THE EXECUTION DIAGNOSTICS

Three completed-task comparisons show how the diagnostics relate to recorded calls (Table 5). The budget and directory examples use one Qwen3-8B checkpoint trained on LTP trajectories with the bank context removed (training seed 42, learning rate 5×10^{-6}, epoch 2); the ticket example compares the selected DeepSeek-V4-Flash-0731 LTP run with the adapted Reflexion run. The cases were selected after inspecting outcomes.

Table 5: **Diagnostic changes have concrete causes in the recorded calls.** Three BFCL tasks; each tuple is (R, P, B), arrows follow the execution order in the second column, and IDs omit the `multi_turn_` prefix.

Task	Executions and official outcomes	(R, P, B)
`miss_param_150`	Plain fail → LTP pass	$(1, .5, .5) \rightarrow (1, 1, 1)$
`miss_param_22`	Plain pass → LTP fail	$(.75, .75, .75) \rightarrow (.5, .5, .5)$
`long_context_23`	Reflexion fail → LTP pass	$(\frac{2}{3}, \frac{2}{3}, \frac{2}{3}) \rightarrow (1, 1, 1)$

In `miss_param_150`, both executions obtain the flight fare and the exchange rate, but Plain sets the budget to the unconverted 15,400 while LTP converts 15,400 GBP to 22,000 USD and passes that value; R stays 1 and P and B rise from .5 to 1.

In `miss_param_22`, both copy the file, but LTP then changes into the archive directory, where the comparison fails on a missing file; the comparison requirement loses its witness and all three scores fall by .25 with a pass-to-fail outcome.

In `long_context_23`, Reflexion reads ticket 7423 but never resolves it, whereas LTP calls `resolve_ticket`; coverage is $2/3$ versus 1.

B.2 SELECTED LTP RUN AND ALTERNATIVE INFERENCE TRAJECTORIES

Table 6 lists three tasks on which the selected LTP run succeeds and the named alternatives fail; the baseline adaptations use a different inference seed, and EasyTool comes from its separate run. In `miss_param_109`, LTP waits for the missing quantity before placing the order and then inspects and cancels the returned order, whereas P2P places an unsolicited 10-share order first and Reflexion retrieves an old order instead of buying; P2P's later correction still earns $(1, 1, 1)$, so coverage does not record every premature action.

In `miss_func_189`, LTP books the flight once `book_flight` becomes available, whereas P2P and EasyTool call `verify_traveler_information` with invented passport data and never book.

In `base_54`, both LTP and P2P receive an unlocked-doors error from `startEngine`; LTP locks the doors, presses the brake fully, and starts the engine, while P2P moves on with the engine stopped. LTP succeeds through recovery from observed errors; its trajectory is not error-free.

Table 6: **Where LTP succeeds and the named alternatives fail.** Three BFCL tasks with (R, P, B) for every displayed execution; IDs omit `multi_turn_`.

Task	LTP (R, P, B)	Comparator (R, P, B)
`miss_param_109`	$(1, 1, 1)$	P2P $(1, 1, 1)$; Reflexion $(\frac{4}{7}, \frac{4}{7}, \frac{3}{7})$
`miss_func_189`	$(1, 1, 1)$	P2P and EasyTool $(.75, .75, .75)$
`base_54`	$(.75, .75, .5)$	P2P $(.5, .5, .25)$

B.3 MATCHED SFT TRAJECTORY COMPARISONS

Three BFCL cases compare Qwen3-8B checkpoints trained on Plain data and on LTP data with the bank context retained, at the same training seed (42), learning rate (2×10^{-6}), and epoch (3), both evaluated with Plain inference and no retrieved bank; the untrained model is a reference from an earlier inference configuration. The Retained SFT checkpoint succeeds in all three cases, and the Plain SFT checkpoint and the untrained model fail (Table 7). The cases were selected after inspecting outcomes.

Table 7: **Retained SFT completes the steps the other students skip.** Correctness and diagnostics for the same three tasks under Plain inference. The case IDs are `multi_turn_miss_param_150`, `multi_turn_long_context_24`, and `multi_turn_long_context_86`.

Case	Training data	Correct	R	P	B
Budget	Untrained	No	.500	.500	.500
	Plain SFT	No	.500	.500	.000
	Retained SFT	Yes	1.000	1.000	1.000
Move and rename	Untrained	No	.667	.500	.500
	Plain SFT	No	.833	.500	.333
	Retained SFT	Yes	.833	.833	.667
Vehicle workflow	Untrained	No	.857	.857	.714
	Plain SFT	No	.714	.714	.714
	Retained SFT	Yes	1.000	1.000	1.000

In the budget case, the Retained SFT model converts 15,400 GBP to 22,000 USD and passes the converted value to the budget setter, after recovering from an initial invalid airport-code guess, whereas the Plain SFT and untrained models compute the conversion but never set the budget.

In the move-and-rename case, the Retained SFT model moves the file, enters the archive, and renames it, whereas the Plain SFT model repeats the move and leaves the new name unapplied, and the untrained model puts a path in the destination and is rejected.

In the vehicle case, the Retained SFT model engages the parking brake, corrects a half brake press to a full one after the returned error, starts the engine, and completes the navigation and tweet steps, whereas the Plain SFT model also recovers the engine start but never engages the parking brake or posts the tweet, and the untrained model abandons the engine recovery after the first brake error and navigates to a substituted destination. These are learned behaviors under Plain inference; no evidence is retrieved at test time.

C Reference results with the official-API DeepSeek-V4.1-Flash

The main text reports every DeepSeek result with the 0731 API version. Earlier runs used the official-API DeepSeek-V4.1-Flash; Table 8 keeps its Test400 diagnostics for reference, together with the Plain-inference AUROC on all 800 tasks for the two Figure 4 models that Table 1 omits (three Plain runs each). With DeepSeek-V4.1-Flash, LTP solved 31 of 49 ACEBench Agent tasks against 27 for Plain, and 56 of 78 ToolTalk conversations against 53.

Table 8: Reference diagnostics not used in the main text. DeepSeek-V4.1-Flash results use Test400; Qwen3.8-27B and GLM-5.3 AUROCs use all 800 tasks, averaged over three runs.

Model	Quantity	R	P	B
DeepSeek-V4.1-Flash	Plain AUROC (397 valid tasks)	0.704	0.715	0.740
	Mean Δ on failure $\rightarrow$ success tasks	$+0.105$	$+0.145$	$+0.068$
	Mean Δ on success $\rightarrow$ failure tasks	-0.149	-0.151	-0.096
Qwen3.8-27B	Plain AUROC (795, 796, 791 valid tasks)	0.717	0.734	0.665
GLM-5.3	Plain AUROC (798 valid tasks per run)	0.660	0.691	0.619