EVOMCP: CO-EVOLVING MCP AGENTS AND TRAINING DATA

Anonymous authors
Paper under double-blind review

ABSTRACT

Supervised fine-tuning of Model Context Protocol (MCP) agents is typically conducted on a static data distribution, which fails to adapt to the student's evolving capabilities and leaves newly exposed weaknesses insufficiently addressed. To address these challenges, we propose EvoMCP, an agent–data mutual evolution framework that enables MCP agents to improve through a closed loop of task evolution, trajectory evolution, and supervised fine-tuning. Specifically, EvoMCP assesses the student's established capabilities and capability gaps from rollout feedback to identify learning targets. Task evolution then translates these targets into executable tasks by adapting the instructions and observable environment states of compatible verified seed tasks while keeping their verifiers aligned. To provide reliable and efficient demonstrations for these tasks, trajectory evolution uses reusable skills to recover failed teacher rollouts and refine successful ones. The resulting task–trajectory pairs are used to fine-tune the student, while accepted tasks seed subsequent evolution. To close the loop, we re-evaluate the updated student and use its feedback to guide the next round of data evolution. Experiments on Toolathlon, MCPMark, and MCP-Atlas demonstrate consistent improvements over the initial student, yielding absolute gains of 16.5%, 10.2%, and 12.6%, respectively.

1 INTRODUCTION

Large language model (LLM) agents handle multi-step workflows that involve finding suitable tools, invoking them with appropriate parameters, responding to tool failures, and combining their outputs into accurate, context-aware responses (Xu et al., 2025). Recent advances, including the Model Context Protocol (MCP), simplify tool integration through standardized interfaces, connecting LLMs to real-world environments and making it easier for agents to discover, invoke, and execute external tools (Xu et al., 2025; Wang et al., 2026). To develop these capabilities, models are trained on pre-collected trajectories using supervised fine-tuning (SFT) (Xu et al., 2025; He et al., 2026). An instance of this training data comprises a task–trajectory pair, where trajectories capture sequences of planning, tool calls, tool responses, and the final model response (Xu et al., 2025).

Existing MCP data pipelines synthesize tasks and tool-use trajectories for agent training (Xu et al., 2025; Wang et al., 2026; Lu et al., 2026; Xu et al., 2026). While synthetic data reduces reliance on human annotation, a fixed training corpus cannot adapt to the student's evolving capabilities, making it difficult to target specific weaknesses or support continual improvement (Yang et al., 2026). For MCP agents trained with SFT, adapting the training data requires executable tasks paired with reliable teacher demonstrations.

Constructing these task–trajectory pairs requires keeping the task instruction, environment, and verifier mutually consistent (Li et al., 2026b; Song et al., 2026). For example, an MCP agent cannot detect declining quiz scores without access to current and historical records. The task instruction and verifier must also agree on which declines should be flagged. Once these tasks are validated, teacher rollouts provide demonstrations for student training. Training on these demonstrations resembles hard distillation in language model training, where a teacher model generates outputs and a student model learns to imitate them (He et al., 2026). A teacher may fail to complete an executable task. However, even successful teacher rollouts can contain incorrect or suboptimal actions (He et al., 2026). This underscores the need for more principled strategies to transform raw rollout trajectories into high-quality training data.

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.

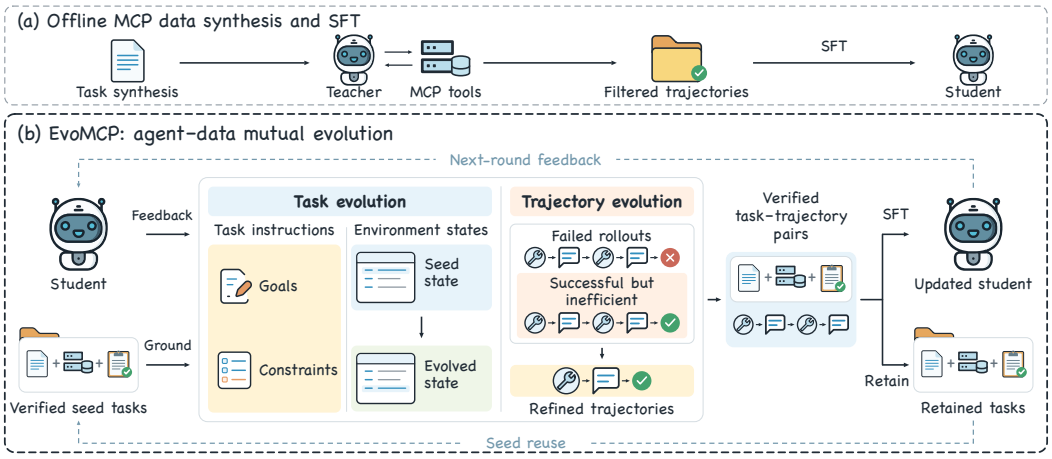


Figure 1: **Self-evolving training in real MCP environments.** (a) TOUCAN synthesizes tasks and filters teacher trajectories executed through real MCP tools for SFT (Xu et al., 2025). (b) EvoMCP uses student rollout feedback to assess capabilities and select learning targets and verified seed tasks. Task evolution adapts task instructions, environment states, or both while keeping verifiers aligned. Trajectory evolution uses reusable skills to recover failed teacher rollouts and refine successful demonstrations. Accepted demonstrations fine-tune the student, and accepted tasks supply seeds for later rounds.

To address these challenges, we propose EvoMCP, an agent–data mutual evolution framework that enables MCP agents to improve through a closed loop of task evolution, trajectory evolution, and supervised fine-tuning. Our core idea is to use student rollout feedback to assess established capabilities and capability gaps, identify learning targets, and guide task evolution accordingly. We therefore evolve complete task–trajectory pairs, adapting compatible verified seed tasks to these targets and improving the teacher demonstrations used to train the student.

As shown in Figure 1, EvoMCP couples task evolution with trajectory evolution to adapt both training tasks and teacher demonstrations. Within task evolution, instruction evolution revises task goals or strengthens constraints. Environment evolution provides a complementary transformation by varying observable execution states, such as data values and object relations, while preserving the capability goal and MCP tool interfaces. These transformations can be used separately or together, with instructions, environments, and verifiers kept mutually consistent. To provide reliable and efficient supervision for these tasks, trajectory evolution uses reusable skills to recover failed teacher rollouts and refine successful ones. Together, these steps allow us to transform seed tasks and teacher rollouts into high-quality training data, enabling more reliable supervised fine-tuning of MCP agents.

After validating the resulting task–trajectory pairs, we use accepted demonstrations to fine-tune the student, and the corresponding tasks become seed tasks for subsequent evolution. The updated student is then re-evaluated to provide fresh feedback for selecting learning targets and compatible seeds in the next round. This closes the loop between data evolution and student learning, allowing the training distribution to adapt as the student's capabilities change. We evaluate EvoMCP on three representative benchmarks, Toolathlon (Li et al., 2026a), MCPMark (Wu et al., 2026), and MCP-Atlas (Bandi et al., 2026), using Qwen3.6-27B as the backbone. By continuously synthesizing new tasks from training-time feedback and refining teacher demonstrations, EvoMCP achieves absolute gains of 16.5%, 10.2%, and 12.6% over the initial student, respectively, demonstrating the effectiveness of agent–data mutual evolution across MCP environments. Our contributions can be summarized as follows:

- We propose EvoMCP, an agent–data mutual evolution framework that alternates between student fine-tuning and the evolution of complete MCP task–trajectory pairs, reusing verified tasks as seeds across rounds.
- Unlike static data synthesis, we incorporate student rollout feedback into learning-target selection and task evolution from verified seeds, and use reusable skills to recover failed teacher rollouts and refine successful demonstrations.

- EvoMCP yields consistent gains over the initial student and static verified SFT across three MCP benchmarks, demonstrating its effectiveness in complex tool-use environments.

2 Related Work

Task and environment synthesis. MCP data pipelines synthesize tasks and collect tool-use trajectories through real MCP servers (Xu et al., 2025; Wang et al., 2026). For task construction, AgentGen (Hu et al., 2025) grounds generation in environment descriptions, while TaskCraft (Shi et al., 2025) composes simpler tasks into complex workflows. EnvScaler (Song et al., 2026) and CLI-Universe (Hua et al., 2026) jointly synthesize executable environments, tasks, and validators. RST (Li et al., 2026b) realigns the components of terminal tasks and reuses validated tasks as seeds across synthesis generations. Our work builds on this line by evolving complete MCP tasks from verified seeds through complementary changes to task instructions and observable environment states, keeping their verifiers aligned.

Feedback-driven agent training. Feedback-driven methods adapt the training distribution as the agent changes. Tool-R0 (Acikgoz et al., 2026) and GenEnv (Guo et al., 2025) target learner-relative difficulty; LoopTool (Zhang et al., 2026) generates data around diagnosed tool-call errors, while CoEvolve (Yang et al., 2026) uses forgetting and uncertainty from rollouts to guide task synthesis. Agent-World (Dong et al., 2026) uses capability diagnosis to guide task and environment expansion. In EvoMCP, student rollout feedback selects learning targets and compatible verified seeds to determine which tasks to reuse and which new tasks to construct.

Trajectory synthesis and refinement. Teacher failures limit demonstration coverage, while errors in successful rollouts affect supervision quality. Trial and Error (Song et al., 2024) forms preference pairs from successful and failed attempts, while STeCa (Wang et al., 2025) calibrates suboptimal actions through step-level feedback. For successful trajectories, WebSTAR (He et al., 2026) masks low-quality steps in the SFT loss. Complementary approaches retain reflections, insights, or skills to guide later attempts, as in Reflexion (Shinn et al., 2023), ExpeL (Zhao et al., 2024), and Ace-Skill (Xiong et al., 2026). EvoMCP uses reusable skills to recover failed teacher rollouts on fixed tasks and refine successful demonstrations to reduce tool errors and avoidable work. Verified demonstrations then fine-tune the student, while accepted tasks become seeds for later evolution. Updated student feedback guides the next round.

3 Problem Formulation

MCP tasks and training data. An MCP task $x = (q, E, V)$ consists of a task instruction q specifying the goal and constraints, an execution environment E with an initial state and available MCP tools, and a task verifier V. To solve the task, an agent interacts with E according to q, producing a trajectory τ of messages, tool calls, and observations. The verifier returns $V(\tau) = 1$ if the execution satisfies the task's success criteria and 0 otherwise. For training, we use accepted task–trajectory pairs $d = (x, \tau)$, where x passes the applicable task checks and τ is a teacher demonstration satisfying $V(\tau) = 1$.

Student feedback. EvoMCP alternates data construction and student training over evolution rounds. At the start of round t, the student policy π_t performs rollouts on a set of MCP tasks. The resulting trajectories and task outcomes yield feedback signals $\mathcal{S}_t$ that characterize the student's established capabilities and capability gaps. This assessment identifies behaviors to reinforce or improve, which become the learning targets for data construction. We then select verified seed tasks whose requirements, tools, and resources support these targets.

Agent–data mutual evolution. At evolution round t, we maintain an accumulated dataset $\mathcal{D}_t$ of verified task–trajectory pairs. The initial dataset $\mathcal{D}_0$ contains verified seed tasks and their teacher demonstrations. Using the learning targets and seed tasks selected above, EvoMCP constructs the training set $\mathcal{D}_t^{\text{train}}$ through task and trajectory evolution. We complete the round by using the selected

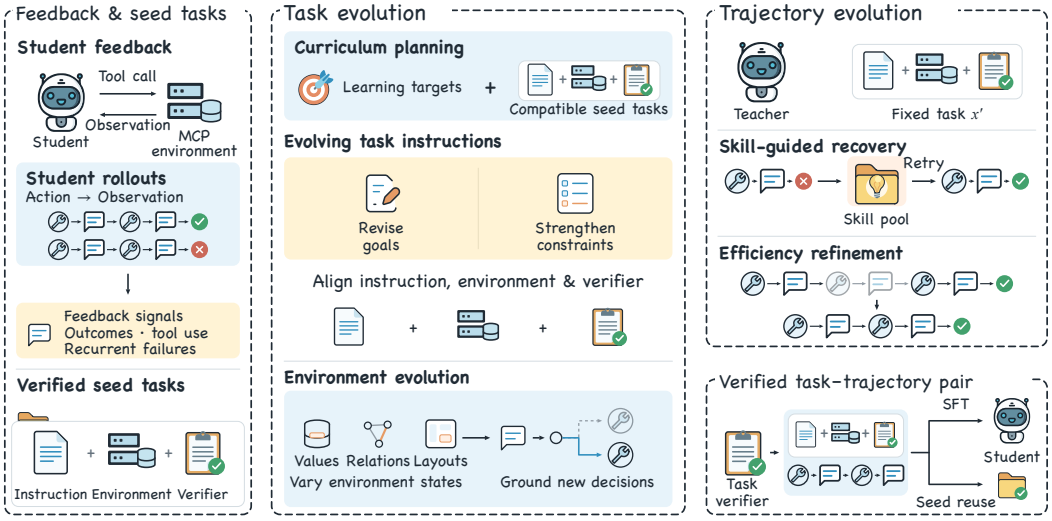


Figure 2: **Overview of EvoMCP.** A curriculum planner uses student feedback to select learning targets and verified seed tasks. Task evolution adapts task instructions, environment states, or both while keeping their verifiers aligned. Trajectory evolution uses reusable skills to recover failed teacher rollouts and refine successful demonstrations on each fixed task. Accepted demonstrations train the student, and the corresponding tasks become seed tasks for subsequent rounds.

demonstrations for SFT and adding newly accepted pairs to $\mathcal{D}_t$, giving

$$\begin{aligned} \pi_{t+1} &= \mathrm{SFT}(\pi_t, \mathcal{D}_t^{\mathrm{train}}), \\ \mathcal{D}_{t+1} &= \mathcal{D}_t \cup \mathcal{D}_t^{\mathrm{train}}. \end{aligned} \tag{1}$$

Re-evaluating the updated student π_{t+1} yields fresh feedback for selecting learning targets in the next round. The expanded dataset $\mathcal{D}_{t+1}$ supplies verified seed tasks for these targets, so subsequent data construction is guided by both the updated student and the accumulated data.

4 Methodology

As shown in Figure 2, EvoMCP couples student-guided task evolution with trajectory evolution to adapt both training tasks and teacher demonstrations. Our core idea is to evolve complete task–trajectory pairs by adapting verified seed tasks to learning targets identified from student rollout feedback and improving the teacher demonstrations used for SFT. Task evolution adapts task instructions, environment states, or both while keeping verifiers aligned. We collect teacher rollouts on these evolved tasks to provide demonstrations for SFT. Trajectory evolution then uses reusable skills to recover failed teacher rollouts and refine successful demonstrations. Accepted demonstrations fine-tune the student, while the corresponding tasks seed later rounds.

4.1 Student-Guided Task Evolution

We assess the student using verifier outcomes, tool-use patterns, and recurrent failures from its rollouts (Yang et al., 2026). The resulting profile identifies established behaviors and capability gaps, linking observed weaknesses to failure mechanisms and supporting rollouts. A curriculum planner uses this profile to select learning targets for retaining learned behaviors, correcting weaknesses, or exploring behaviors with limited evidence. It matches these targets to verified seed tasks whose requirements, tools, and resources support the intended behavior, together with their verified demonstrations from $\mathcal{D}_t$. The planner may reuse an existing pair or evolve its task. For evolution, it selects the direction of change, while a task generator determines the concrete requirements and state changes supported by the selected seed tasks.

When evolving the task from a single verified seed pair $d = (x, \tau) \in \mathcal{D}_t$, the learning target derived from feedback $s \in \mathcal{S}_t$ guides two complementary transformations of $x = (q, E, V)$.

$$\begin{aligned} \mathcal{T}_s(x) &= (q', E_T, V_T), \\ \mathcal{E}_m(x) &= (q_m, E_m, V_m). \end{aligned} \tag{2}$$

For a selected learning target, $\mathcal{T}_s$ changes task requirements, while $\mathcal{E}_m$ produces task instances with varied observable states. Here, m specifies the state elements to vary, such as data values or object relations. The planner can select either route or apply a compatible environment transformation to the task produced by $\mathcal{T}_s$. We denote the resulting task by x'.

Evolving task instructions. Conditioned on the selected learning target and compatible verified seed tasks, we prompt the task generator to specify the behavior to introduce and the requirements to retain. Specifically, it revises task goals or strengthens constraints based on the tools and resources of the selected seed tasks. For example, a goal revision can combine subgoals and introduce dependencies between them. The framework then generates q' and updates resources in E_T as needed, binding task references to information accessible through tool interactions.

Accordingly, we retain V_T when the success criteria are unchanged, and extend or revise its checks when requirements are added or the objective changes. We then check consistency among q', E_T, and V_T. Together with evidence from the verified seed tasks, resource checks and positive and negative verifier tests support solvability assessment before teacher collection. This process yields tasks tailored to the selected learning targets, with aligned instructions, environments, and verifiers.

Environment evolution. In addition to changing task requirements, we evolve environment states to provide diverse task instances for the selected learning targets. The motivation is that the same capability can require different information, tool arguments, or decisions under different observable conditions. For example, we vary task-relevant data values, object relations, or layouts.

To keep each instance consistent, we update references in q_m and the factual and resource bindings of V_m as needed to match E_m, while preserving the parent task's success criteria. We collect teacher demonstrations for the selected learning targets across these varied observable states.

4.2 Trajectory Evolution

Task validity alone does not ensure a useful demonstration: teacher rollouts may fail, while successful ones can contain suboptimal intermediate actions that full-trajectory SFT directly supervises (He et al., 2026). We recover failed rollouts and refine successful ones while keeping task x' fixed.

Skill-guided failure recovery. We first distinguish model-behavior failures from task, verifier, and runtime failures. Only model-behavior failures enter skill-guided recovery. For such failures, EvoMCP routes a compatible active skill. If none applies, it distills a new recovery skill from the failed rollout and agent-visible feedback. Each skill encodes an observable failure condition, a corrective procedure grounded in agent-visible information, and a completion check. The teacher retries in a fresh session, and only verifier-passing trajectories are retained. Newly distilled skills enter the pool only after passing the unchanged verifier on their source task.

Efficiency refinement. Successful trajectories are audited for real tool errors and avoidable work. Let $N_{\mathrm{red}}(\tau)$ and $N_{\mathrm{succ}}(\tau)$ denote redundant and all successful tool calls. For assistant segment j, let a_j be its avoidable fraction and n_j its length. We define the *avoidable-work ratio* (AWR) as

$$\mathrm{AWR}(\tau) = \max\left\{ \frac{N_{\mathrm{red}}(\tau)}{\max\{N_{\mathrm{succ}}(\tau), 1\}}, \frac{\sum_j n_j a_j}{\max\{\sum_j n_j, 1\}} \right\}. \tag{3}$$

If a trajectory misses its task-specific tool-error or AWR threshold, the teacher reruns the unchanged task using the audit findings as teacher-only guidance. Among verifier-passing candidates, we prioritize fewer tool errors and then lower AWR.

4.3 Validation, Training, and Reuse

Before SFT, we exclude teacher-only skill guidance and judge annotations from the training examples so that the training context contains the task inputs available to the student. For an evolved task x', the selected and cleaned trajectory $\tau^\star$ forms a candidate pair $d' = (x', \tau^\star)$. It enters $\mathcal{D}_t^{\text{train}}$ only if the task passes validation checks and the teacher execution has passed the task verifier. We also check that the exported messages preserve a consistent sequence of tool calls and observations. Verified task–trajectory pairs selected for direct reuse enter $\mathcal{D}_t^{\text{train}}$ with their existing demonstrations.

Following Equation 1, demonstrations in $\mathcal{D}_t^{\text{train}}$ supervise the update from π_t to π_{t+1}, and newly accepted task–trajectory pairs expand $\mathcal{D}_{t+1}$. Re-evaluating the updated student through the same execution framework yields fresh feedback that, together with the expanded seed pool, guides task selection and evolution in the next round.

5 Experiments

5.1 Experimental Setup

Benchmarks and metrics. We evaluate on Toolathlon (Li et al., 2026a), MCPMark (Wu et al., 2026), and MCP-Atlas (Bandi et al., 2026). They cover long-horizon, stateful tool use, cross-server orchestration, and claim-level synthesis. We report Toolathlon task success and MCPMark and MCP-Atlas Pass@1, using the official 0.75 claim-coverage threshold for the latter.

Data and evaluation. Verified seed task–trajectory pairs initialize $\mathcal{D}_0$. Benchmark rollouts provide student feedback for learning-target and seed-task selection, and successive checkpoints are evaluated on the same benchmark task sets. We refresh this feedback after each student update. All student checkpoints use the same evaluation setup. For MCP-Atlas, we use GLM-5.2 as the claim-level judge, with its prompt and decoding settings held fixed.

Models and training. We use Qwen3.6-27B as the student and Qwen3.8-Max as the task generator and teacher. R0 denotes the initial student, and R1–R3 denote the checkpoints after one to three rounds of full-parameter SFT. We train for three epochs on 2,400 verifier-accepted demonstrations per round. We use a learning rate of 2×10^{-6} and a global batch size of 128. Further implementation details and results appear in the appendix.

5.2 Main Results

Table 1 compares public reference models with the initial student and successive EvoMCP checkpoints. Public-model scores and parameter counts follow official reports and model cards (DeepSeek-AI et al., 2026), using their source-specific harnesses and inference budgets.

Overall performance. EvoMCP consistently improves student performance across all three benchmarks. After three rounds, it reaches 53.4% on Toolathlon, 66.9% on MCPMark, and 73.6% on MCP-Atlas, yielding gains of 16.5%, 10.2%, and 12.6% over R0. Performance improves in every round on all three benchmarks, showing that successive rounds of data evolution continue to benefit the updated student. We next compare against continued training on fixed seed data to examine the contribution of evolving the training pairs.

5.3 Ablation Study

Comparison settings. We compare EvoMCP with *Static SFT*. Starting from R0, this baseline repeatedly trains on the same verifier-passing seed-task demonstrations. At R1, R2, and R3, its cumulative training steps match those of the corresponding EvoMCP checkpoint.

At R1, we further remove one component at a time. *Without student feedback*, learning-target selection, seed selection, transformation planning, and task construction exclude student outcomes and capability diagnoses, relying on seed semantics, available resources, and task diversity. *Without task evolution*, the selected verified tasks are reused unchanged. *Without trajectory evolution*, we retain the first verifier-passing teacher rollout without recovery or refinement. All variants use the

Table 1: **Performance on three tool-use benchmarks (%).** Scores are task success for Toolathlon and Pass@1 for MCPMark and MCP-Atlas. Bold and underlined values mark the highest and second-highest reported scores per benchmark, respectively. Blue rows denote EvoMCP checkpoints. Params gives total/activated parameters. Public reference scores use their original evaluation settings.

Model	Params	Toolathlon	MCPMark	MCP-Atlas
Closed-source agents				
GPT-5.4 (xhigh)	–	**54.6**	62.5	70.6
Claude Opus 4.5	–	43.5	42.6	69.8
Gemini 3 Pro	–	36.4	53.9	70.3
Open-weight models				
Kimi K2.5	1T/32B	27.8	29.5	64.4
DeepSeek-V4-Flash	284B/13B	47.8	63.8	69.0
Student checkpoints				
Qwen3.6-27B	27B/27B	36.9	56.7	61.0
+ EvoMCP(R1)	27B/27B	46.6	60.6	68.4
+ EvoMCP(R2)	27B/27B	50.5	<u>64.5</u>	<u>71.3</u>
+ EvoMCP(R3)	27B/27B	<u>53.4</u>	**66.9**	**73.6**

Table 2: **Component ablations at R1 and Static SFT comparisons through R3.** Cumulative training steps match per round. Scores (%) use the metrics in Table 1. Checkmarks indicate enabled components: student feedback (Stud. fb.), task evolution (Task evo.), and trajectory evolution (Traj. evo.). Bold marks each round's best scores, and shading denotes EvoMCP.

Variant	Stud. fb.	Task evo.	Traj. evo.	Toolathlon	MCPMark	MCP-Atlas
Initial student (R0)	–	–	–	36.9	56.7	61.0
Round 1						
Static SFT	×	×	×	40.8	57.4	65.8
w/o student feedback	×	✓	✓	42.2	58.2	66.4
w/o task evolution	✓	×	✓	42.4	57.4	66.2
w/o trajectory evolution	✓	✓	×	44.3	59.5	67.8
EvoMCP	✓	✓	✓	**46.6**	**60.6**	**68.4**
Round 2						
Static SFT	×	×	×	40.8	57.4	67.2
EvoMCP	✓	✓	✓	**50.5**	**64.5**	**71.3**
Round 3						
Static SFT	×	×	×	39.8	58.2	67.2
EvoMCP	✓	✓	✓	**53.4**	**66.9**	**73.6**

same initial checkpoint, number of demonstrations, and SFT settings. The remaining components follow the full method.

Effect of iterative evolution. Table 2 shows that EvoMCP outperforms Static SFT at every round. Its absolute gains grow from 5.8%, 3.2%, and 2.6% at R1 to 13.6%, 8.7%, and 6.4% at R3 on Toolathlon, MCPMark, and MCP-Atlas, respectively. With cumulative training steps matched, these results support refreshing the task–trajectory pairs as the student improves, extending the benefit of data evolution beyond the first round.

Contribution of individual components. At R1, removing any component reduces scores on all three benchmarks. The larger drops follow the removal of student feedback (2.0–4.4 points) or task evolution (2.2–4.2 points), highlighting the value of selecting learning targets from student behavior and constructing tasks to realize them. Removing trajectory evolution also lowers scores by 0.6–2.3

Table 3: **Effects of task and trajectory evolution.** (a) Seed–evolved task pairs, with Δ denoting evolved-minus-seed differences in percentage points. (b) Matched verifier-passing teacher demonstrations before and after refinement. AWR denotes avoidable-work ratio. Down arrows indicate lower is better. Shading and boldface identify the evolved condition.

(a) Task evolution

Metric	Seed	Evolved	Δ
Target realization (%)	27.8	**85.7**	+57.9
Behavior-changing tasks (%)	–	**100.0**	–
Student failure rate (%)	24.5	**45.3**	+20.8
Teacher success rate (%)	89.7	**78.9**	−10.8

(b) Trajectory evolution

Metric	w/o Traj. Evo.	w/ Traj. Evo.
Mean tool-call errors ↓	2.6	**1.6**
Mean AWR (%) ↓	31.1	**30.6**
Mean output tokens ↓	9,752.0	**8,873.0**

points, showing a further benefit from recovering and refining teacher demonstrations. Together, these results support combining feedback-guided task construction with trajectory evolution.

5.4 Analysis of Task and Trajectory Evolution

Effect of task evolution. Table 3(a) compares seed and evolved tasks at R1 against the same feedback-derived learning target for each pair. Target realization checks whether the intended behavior is required by the task, supported by the environment, and tested by the verifier. Behavior change measures whether evolution alters the required actions or decisions. We also evaluate student failure and teacher success using R0 and the same teacher under matched execution budgets, before recovery or refinement.

Task evolution increases target realization from 27.8% to 85.7%, and all sampled evolved tasks require changed solution behavior. It also raises R0's failure rate from 24.5% to 45.3%, while the teacher still succeeds on 78.9% of evolved tasks, compared with 89.7% of seed tasks. Together, these results show that evolved tasks more consistently instantiate the selected learning targets, expose unmet demands for the student, and remain solvable by the teacher in most cases. This connects student feedback to executable tasks for collecting successful teacher demonstrations.

For example, an evolved analyst-rating task requires pooling observations across three forecast horizons before computing a hit rate, where the seed reports each horizon separately. Its updated verifier checks the pooled statistics, observation counts, and report structure, and the collected teacher demonstration passes. This case illustrates how a numeric-accuracy target changes the required aggregation while reusing the source data.

Effect of trajectory evolution. Table 3(b) evaluates trajectory refinement at R1 on matched verifier-passing teacher demonstrations. Refinement reduces mean tool-call errors from 2.6 to 1.6 and mean output tokens from 9,752.0 to 8,873.0, a 9.0% reduction. Mean AWR changes more modestly, from 31.1% to 30.6%. These results show that refinement yields more concise successful demonstrations with fewer tool-call errors. The gains in Table 2 further support the combined trajectory-evolution stage for student SFT.

Evolution dynamics. Figure 3 tracks student performance and training-data changes across rounds. In panel (b), task validity increases from 78.0% at R1 to 88.0% at R3, trajectory acceptance from 72.0% to 79.0%, and training-pair admission from 68.0% to 73.0%. The improvement extends from task construction through teacher collection to the admission of verified training pairs. Thus, later rounds yield a higher proportion of usable supervision alongside the continued student gains in panel

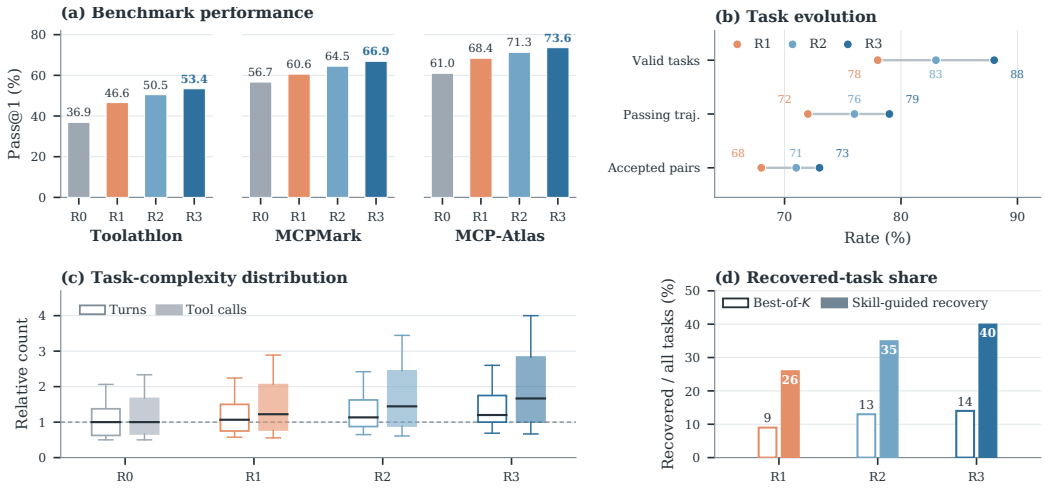


Figure 3: **Evolution dynamics across rounds.** (a) Benchmark performance from R0 to R3. (b) Task-validity, trajectory-acceptance, and training-pair-admission rates. (c) Turns and tool calls, each normalized by its R0 median, as proxies for interaction workload. (d) Recovered tasks as a percentage of all tasks in each round under a matched teacher-rollout budget. Best-of-K denotes unguided retries.

(a). These trends complement the gains over Static SFT in Table 2, where cumulative training steps are matched at each checkpoint.

Panel (c) shows how the teacher's interaction workload changes. Both the median and upper quartile of tool-call counts increase, while turns grow more moderately. The distribution therefore shifts toward executions with more tool interaction, extending the change beyond a few unusually long trajectories. Together with panel (b), this shows that the pipeline continues to produce usable training pairs as teacher executions become longer and involve more tool use.

Panel (d) compares skill-guided recovery with unguided retries under a matched teacher-rollout budget. The recovered-task share grows from 26.0% at R1 to 40.0% at R3 with skill guidance, compared with 9.0% to 14.0% for unguided retries. The absolute advantage therefore widens from 17.0% to 26.0% across rounds. This advantage is present from the first round and persists as the task pool evolves, supporting reusable recovery guidance for obtaining demonstrations from initially unsuccessful executions. Alongside Table 3(b), these results highlight the two roles of trajectory evolution: recovering successful demonstrations for initially failed tasks and refining the efficiency of already successful demonstrations. Together, task adaptation and teacher-demonstration improvement produce verified experience for both student training and subsequent task construction, linking the gains within each round to continued data evolution across rounds.

6 Conclusion

We introduce EvoMCP, a framework for co-evolving MCP agents and verified task–trajectory pairs through supervised fine-tuning. By using student feedback to guide task construction and improving teacher demonstrations through recovery and refinement, EvoMCP adapts its training data across rounds. Accepted demonstrations train the student, while accepted tasks seed subsequent evolution. Experiments on three tool-use benchmarks show consistent gains across three rounds and a widening advantage over Static SFT under matched cumulative training steps. Component ablations further support the roles of student feedback, task evolution, and trajectory evolution. Together, these findings highlight task construction and teacher supervision as complementary components of iterative MCP-agent training, with verified experience connecting data evolution and student learning.

AI use statement. Generative AI tools were used to assist with language editing and improving the readability of the manuscript. All AI-assisted content was reviewed and verified by the authors. The authors take full responsibility for the final content of this work.

REFERENCES

Emre Can Acikgoz, Cheng Qian, Jonas Hübotter, Heng Ji, Dilek Hakkani-Tür, and Gokhan Tur. Tool-r0: Self-evolving llm agents for tool-learning from zero data, 2026. URL https://arxiv.org/abs/2602.21320.

Chaithanya Bandi, Razvan-Gabriel Dumitru, Ben Hertzberg, Divyansh Agarwal, Geobio Boo, Tejas Polakam, Sami Hassaan, Jeff Da, HiJae Kim, Vipul Gupta, Manasi Sharma, Andrew Park, Martin Dimakis, Ernesto Gabriel Hernandez Montoya, Dan Rambado, Ivan Salazar, Rafael Cruz, MohammadHossein Rezaei, Chetan Rane, Ben Levin, Daniel Yue Zhang, Brad Kenstler, and Bing Liu. Mcp-atlas: A large-scale benchmark for tool-use competency with real mcp servers, 2026. URL https://arxiv.org/abs/2602.00933.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang,

Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, Jiajie Jin, Yutao Zhu, Hanbin Wang, Fangyu Lei, Qinyu Luo, Mingyang Chen, Zehui Chen, Jiazhan Feng, Ji-Rong Wen, and Zhicheng Dou. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence, 2026. URL https://arxiv.org/abs/2604.18292.

Jiacheng Guo, Ling Yang, Peter Chen, Qixin Xiao, Yinjie Wang, Xinzhe Juan, Jiahao Qiu, Ke Shen, and Mengdi Wang. Genenv: Difficulty-aligned co-evolution between llm agents and environment simulators, 2025. URL https://arxiv.org/abs/2512.19682.

Yifei He, Pranit Chawla, Yaser Souri, Subhojit Som, and Xia Song. WebSTAR: Scalable data synthesis for computer use agents with step-level filtering. 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)*, pp. 516–533, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.21. URL https://aclanthology.org/2026.acl-long.21/.

Mengkang Hu, Pu Zhao, Can Xu, Qingfeng Sun, Jianguang Lou, Qingwei Lin, Ping Luo, and Saravan Rajmohan. Agentgen: Enhancing planning abilities for large language model based agent via environment and task generation, 2025. URL https://arxiv.org/abs/2408.00764.

Zhanbo Hua, Yifan Yao, Weihao Xie, Yongchi Zhao, Minghao Liu, Ruizhi Qiu, Zhewei Huang, Zun Wang, Yiyan Ji, Yunhai Ye, Letian Zhu, Xinping Lei, Han Li, Zhiyuan Ma, Zili Wang, Zhaoxiang Zhang, and Jiaheng Liu. Cli-universe: Towards verifiable task synthesis engine for terminal agents, 2026. URL https://arxiv.org/abs/2606.22883.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, Junteng Liu, Zhaochen Su, Yiyang Guo, Fan Zhou, Lueyang Zhang, Juan Michelini, Xingyao Wang, Xiang Yue, Shuyan Zhou, Graham Neubig, and Junxian He. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution, 2026a. URL https://arxiv.org/abs/2510.25726.

Zhongzhi Li, Yucheng Shi, Zongxia Li, Ruhan Wang, Anhao Li, Zixun Huang, Junyao Yang, Lei Ke, Ninghao Liu, Haitao Mi, and Leowei Liang. Recursive synthesis for long-horizon terminal tasks, 2026b. URL https://arxiv.org/abs/2608.05466.

Yuxuan Lu, Ziyi Wang, Yingzhou Lu, Yisi Sang, Jiri Gesi, Xianfeng Tang, Yimeng Zhang, Zhenwei Dai, Hui Liu, Hanqing Lu, Chen Luo, Qi He, Benoit Dumoulin, Jing Huang, and Dakuo Wang. Firefly: Illuminating large-scale verified tool-call data generation from real apis, 2026. URL https://arxiv.org/abs/2605.17558.

Dingfeng Shi, Jingyi Cao, Qianben Chen, Weichen Sun, Weizhen Li, Hongxuan Lu, Fangchen Dong, Tianrui Qin, King Zhu, Minghao Liu, Jian Yang, Ge Zhang, Jiaheng Liu, Changwang Zhang, Jun Wang, Yuchen Eleanor Jiang, and Wangchunshu Zhou. Taskcraft: Automated generation of agentic tasks, 2025. URL https://arxiv.org/abs/2506.10055.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Envscaler: Scaling tool-interactive environments for llm agent via programmatic synthesis, 2026. URL https://arxiv.org/abs/2601.05808.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of LLM agents. 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)*, pp. 7584–7600, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.409. URL https://aclanthology.org/2024.acl-long.409/.

Hanlin Wang, Jian Wang, Chak Tou Leong, and Wenjie Li. STeCa: Step-level trajectory calibration for LLM agent learning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 11597–11614, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.604. URL https://aclanthology.org/2025.findings-acl.604/.

WenHao Wang, Peizhi Niu, Zhao Xu, Zhaoyu Chen, Jian Du, Yaxin Du, Xianghe Pang, Keduan Huang, Yanfeng Wang, Qiang Yan, and Siheng Chen. MCP-flow: Facilitating LLM agents to master real-world, diverse and scaling MCP tools. 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)*, pp. 5087–5116, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.231. URL https://aclanthology.org/2026.acl-long.231/.

Zijian Wu, Xiangyan Liu, xinyuan zhang, Lingjun Chen, Fanqing Meng, Lingxiao Du, Yiran Zhao, Fanshi Zhang, Yaoqi Ye, Jiawei Wang, Zirui Wang, Jinjie Ni, Yufan Yang, Arvin Xu, and Michael Qizhe Shieh. Mcpmark: A benchmark for stress-testing realistic and comprehensive mcp use. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 79852–79895, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/8138d211ce8790fdfbeeeb9781838a37-Paper-Conference.pdf.

Feng Xiong, Zengbin Wang, Yong Wang, Xuecai Hu, Jinghan He, Liang Lin, Yuan Liu, and Xiangxiang Chu. Ace-skill: Bootstrapping multimodal agents with prioritized and clustered evolution, 2026. URL https://arxiv.org/abs/2605.08887.

Minrui Xu, Zilin Wang, Mengyi DENG, Zhiwei Li, Zhicheng Yang, Xiao Zhu, Yinhong Liu, Boyu Zhu, Baiyu Huang, Chao Chen, Heyuan Deng, Fei Mi, Lifeng Shang, Xingshan Zeng, and Zhijiang Guo. Envfactory: Scaling tool-use agents via executable environments synthesis and robust rl, 2026. URL https://arxiv.org/abs/2605.18703.

Zhangchen Xu, Adriana Meza Soria, Shawn Tan, Anurag Roy, Ashish Sunil Agrawal, Radha Poovendran, and Rameswar Panda. Toucan: Synthesizing 1.5m tool-agentic data from real-world mcp environments, 2025. URL https://arxiv.org/abs/2510.01179.

Shidong Yang, Ziyu Ma, Tongwen Huang, Yiming Hu, Yong Wang, and Xiangxiang Chu. Co-Evolve: Training LLM agents via agent-data mutual evolution. 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)*, pp. 23015–23036, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1055. URL https://aclanthology.org/2026.acl-long.1055/.

Kangning Zhang, Weiwen Liu, Wenxiang Jiao, Kounianhua Du, Yuan Lu, Weinan Zhang, and Yong Yu. LoopTool: Closing the data–training loop for robust LLM tool calls. 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)*, pp. 21144–21164, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.968. URL https://aclanthology.org/2026.acl-long.968/.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners, 2024. URL https://arxiv.org/abs/2308.10144.

APPENDIX CONTENTS

A Training and Implementation Details 14
- A.1 Training configuration 14
- A.2 MCP-Atlas evaluation protocol 14
- A.3 Task construction and validation 14
- A.4 Failure attribution and skill routing 15
- A.5 Efficiency selection and SFT export 17

B Complete EvoMCP Procedure 17

C Pipeline Prompt Templates 18
- C.1 Curriculum planning and task construction 19
- C.2 Task realization 19
- C.3 Failure attribution 20
- C.4 Recovery-skill distillation 20
- C.5 Trajectory-efficiency assessment 21

D Qualitative Cases 21
- D.1 Task-evolution cases 22
- D.2 Trajectory-evolution case 25

E Additional Experimental Analyses 26
- E.1 Benchmark feedback and evaluation 26
- E.2 Student progress on the feedback pool 26
- E.3 A behavior-level correction 28
- E.4 Task-evolution measurements 28

A Training and Implementation Details

A.1 Training configuration

Table 4 reports the SFT settings used in each evolution round. Static SFT repeatedly uses the same seed demonstrations, matching the cumulative training steps at each checkpoint.

Table 4: Supervised fine-tuning configuration.

Setting	**Qwen3.6-27B**
Training method	Full-parameter SFT
Training trajectories per round	2,400
Epochs per round	3
Optimizer	Adam
Adam betas	$(0.9, 0.95)$
Weight decay	0.1
Learning rate	2×10^{-6}
Minimum learning rate	1×10^{-8}
Schedule	Cosine decay
Warmup ratio	10%
Micro batch size	1
Global batch size	128
Maximum sequence length	131,072
Hardware	128 PPU-810E
Tensor parallelism	4
Context parallelism	2
Data parallelism	16

A.2 MCP-Atlas evaluation protocol

All MCP-Atlas results for our student checkpoints use GLM-5.2 as the claim-level judge under one fixed scoring protocol. Each reference claim is evaluated independently against the student's final answer and assigned one of three outcomes: fulfilled, partially fulfilled, or not fulfilled, mapped to 1, 0.5, and 0, respectively. Claim coverage is the mean score across all reference claims for a task, and the task passes when coverage is at least 0.75.

Judge calls use temperature 0, a maximum output length of 4,096 tokens, and a strict JSON schema containing the claim, outcome, justification, and confidence. Empty or failed student responses receive zero coverage. The judge model, prompt, decoding settings, outcome mapping, and pass threshold are held fixed across all student checkpoints.

A.3 Task construction and validation

The curriculum planner receives the student capability report and the available seed roster. It allocates retain, remediate, and explore lanes, selects compatible runtime seeds, and specifies a mutation axis and change strength. Dispatch turns these choices into individual work orders. The planner selects the learning direction; the task builder determines the concrete requirement change from the selected seeds and their executable materials.

The builder receives the work order, the complete seed request, indexed source and implementation materials, available operations and MCP tools, and previously generated sibling goals. It can inspect missing source facts before returning an operation with its parameters. For a local change, the implementation preserves the seed setup and unaffected evaluator checks, and patches the changed rule and its tests. Typed task programs additionally encode goals, dependencies, constraints, relevant tools, and preserved clauses; their surface realizer produces one coherent request. Public requests state the required outcome without revealing student diagnoses, private answers, or evaluator internals.

Environment evolution operates on a copied initial state. Changed records and resources are bound back to the request, while the MCP interface and unchanged success criteria are preserved. When a state change alters an observable requirement, the corresponding verifier input is updated with the

same binding. This implementation treats the request, state, and verifier as one candidate; none is admitted independently.

Each candidate retains its source seeds, capability-report reference, planner assignment, construction record, and execution receipts. Environment variants used for held-out evaluation are kept outside the curriculum and training export. The cases in Section D.1 illustrate both an environment pair from this evaluation split and a requirement change applied to an evolved training seed.

Validation checks that the initial state materializes, declared tools are available, and task references resolve to observable resources. Positive tests must satisfy the intended requirements, while focused counterexamples that omit or corrupt a changed requirement must fail. Before teacher collection, we assess solvability using evidence from the verified seeds and these positive and negative tests. Inclusion in the training set additionally requires a teacher demonstration that passes the candidate's native verifier.

A.4 Failure attribution and skill routing

A failed teacher rollout is assigned to model behavior, the task or verifier contract, the execution environment, the harness, or an unresolved mixture. Only model-behavior failures enter skill-guided recovery. The diagnosis records the first problematic step, failed requirements, supporting evidence, failure type and subtype, a minimal correction, and confidence. Skill construction requires non-empty evidence and confidence of at least 0.8.

Table 5 lists the behavioral taxonomy used for diagnosis and routing.

Table 5: Trajectory-failure taxonomy used for diagnosis and skill routing.

Type	**Representative subtypes**	**Boundary**
Planning	Turn-budget exhaustion; redundant search; incorrect ordering	Available actions are arranged into an ineffective plan.
Tool selection	Wrong tool; omitted required tool	An exposed operation is omitted or replaced by an unsuitable one.
Argument	Missing argument; wrong scope; invalid schema	The selected tool is appropriate, but its arguments are not.
Recovery	No retry; repeated failed call; ineffective fallback	Visible failure feedback is not followed by an appropriate response.
Evidence	Incomplete retrieval; source substitution; mismatched evidence	Required evidence is retrievable but collected or associated incorrectly.
Synthesis	Deliverable omission; truncation; format or consistency failure	Evidence is available, but the final response is incomplete or inconsistent.

A stored skill contains a trigger, procedure, stop condition, anti-patterns, applicability metadata, and provenance. It may refer only to information visible in the task and trajectory. The router requires the same failure type, prefers an exact subtype match over a wildcard match, checks server compatibility, and injects at most one skill. If no active skill applies, a new candidate is distilled from the failed rollout and visible verifier evidence. It becomes reusable only after a fresh source-task recovery passes the original verifier.

Figures 4 and 5 show two representative recovery skills. The cards omit source-task values while preserving the reusable trigger, procedure, stop condition, anti-patterns, and scope.

Synthesis skill: reconcile ranked top-k and remaining segments

Trigger. Use when grouped counts must be divided into a leading top-k segment and a remaining segment, with totals, percentages, ratios, or averages reconciled in the final answer.

Procedure.

1. Group by the target field, sort the complete ranked list, and inspect all groups before selecting a split point.
2. If k is not fixed by the request, infer a coherent leading segment from the ranking and later references to the remaining groups; do not default to top-3.
3. Compute the leading and remaining totals from the grouped results returned for the current task.
4. Verify that the two segment totals reconcile with an independently observed collection total.
5. Derive every requested comparison, including shares, ratios, segment averages, and the range of the remaining groups.
6. Before answering, check that the selected collection, field validation, ranked groups, segment labels, totals, comparisons, and conclusion are all present.
7. If an earlier calculation used a different split, correct it explicitly rather than leaving contradictory segment definitions.

Stop condition. Stop when the selected segments are correct, their totals match the independent count, and every requested comparison is stated.

Avoid. Defaulting to top-3; listing group counts without segment totals; mixing different values of k; or drawing a conclusion before the numeric reconciliation.

Scope: Teacher-only grouped-data synthesis repair **Status:** Active; source recovery passed

Figure 4: **Ranked-segment reconciliation skill.** The procedure partitions ranked groups, reconciles their totals, and checks every requested comparison without retaining source-task values.

Evidence skill: continue after a transient read rejection

Trigger. Use when a read-only Notion request is rejected or temporarily unavailable while the task still requires evidence from accessible pages.

Procedure.

1. Retry the same read once when the error explicitly indicates a transient or retryable failure.
2. After a database query succeeds, enumerate the visible candidate page identifiers and titles before selecting pages to inspect.
3. For each selected page, request its child blocks and record the returned results, pagination flag, and cursor.
4. If a child-block request is rejected, retry it once when the failure is retryable; otherwise, do not claim that the page was inspected.
5. Continue with another visible candidate page until the required successful inspections are complete or no candidate remains.
6. In the final answer, distinguish successful observations from rejected attempts; only successful calls support claims about page content.
7. Ground database identity, page identity, and block structure in their corresponding visible tool responses.

Stop condition. Stop when all required pages have successful child-block responses, or when every visible candidate has been attempted and the answer explicitly limits its claims to successfully inspected pages.

Avoid. Treating the first rejection as permanent; claiming an empty result after a rejected call; stopping after one page when several are required; inventing page names or outcomes; or retrying a non-retryable failure indefinitely.

Scope: Teacher-only Notion evidence collection **Status:** Active; source recovery passed

Figure 5: **Complete transient-read recovery skill.** This active skill continues evidence collection after a temporary tool rejection while preventing failed calls from being presented as successful observations.

A.5 Efficiency selection and SFT export

The tool-path and assistant-output audits produce the quantities used by AWR in Equation 3. Tool-error and AWR thresholds are set per task rather than globally. Candidates that do not meet their task-specific thresholds are rerun on the unchanged task in a fresh session, with the audit findings supplied as teacher-only refinement guidance. If no additional candidate meets both thresholds, the implementation selects among correct candidates by the ordered key $(e(\tau), \mathrm{AWR}(\tau), k)$, where $e(\tau)$ is the number of real tool errors and k is the generation index. The original correct trajectory is included in this candidate set.

The SFT exporter retains the public request, model-visible reasoning, tool calls, tool observations, and final answer. Routed skills, failure diagnoses, verifier feedback, efficiency judgments, and other teacher-only metadata are removed. The exporter also checks that the call/result sequence is complete and that every invoked tool belongs to the task's recorded interface.

B Complete EvoMCP Procedure

Algorithm 1 gives the round-level procedure that connects the two evolution stages described in Section 4.

Algorithm 1 One round of EvoMCP

Require: Student π_t; verified pool $\mathcal{D}_t$; skill pool $\mathcal{K}_t$; budget B_t
Ensure: Student π_{t+1}; verified pool $\mathcal{D}_{t+1}$; skill pool $\mathcal{K}_{t+1}$

Evaluate π_t and summarize feedback signals $\mathcal{S}_t$
Initialize $\mathcal{D}_t^{\text{train}} \leftarrow \varnothing$ and $\mathcal{K}_{t+1} \leftarrow \mathcal{K}_t$
while budget B_t remains **do**
    Select a learning target and compatible verified seeds from $\mathcal{D}_t$
    Evolve the task instruction, environment, or both to obtain $x' = (q', E', V')$
    if x' fails consistency, materialization, or verifier checks **then**
        reject x' and continue
    end if
    Execute a fresh teacher rollout τ_0 on x'
    if $V'(\tau_0) = 0$ and the failure is attributable to the teacher **then**
        if a compatible active recovery skill exists **then**
            route the existing skill
        else
            distill a source-grounded recovery skill
        end if
        retry in a fresh session on the unchanged task
    end if
    if no teacher trajectory passes V' **then**
        reject the candidate and continue
    end if
    Assess correct candidates for tool errors and avoidable work
    if a candidate misses its task-specific thresholds and budget remains **then**
        rerun on the unchanged task using the audit findings as guidance
        add only verifier-passing refinements to the candidate set
    end if
    Select $\tau^\star$ and remove teacher-only guidance and annotations
    Add $(x', \tau^\star)$ to $\mathcal{D}_t^{\text{train}}$
    Update $\mathcal{K}_{t+1}$ with reusable source-validated skills
end while
$\pi_{t+1} \leftarrow \mathrm{SFT}(\pi_t, \mathcal{D}_t^{\text{train}})$
$\mathcal{D}_{t+1} \leftarrow \mathcal{D}_t \cup \mathcal{D}_t^{\text{train}}$

C Pipeline Prompt Templates

The following cards show instructions for curriculum planning, task construction, task realization, failure attribution, recovery-skill distillation, and efficiency assessment. The planner and builder cards quote excerpts of the code templates; their task-specific inputs are supplied at runtime. Efficiency assessment uses two calls because tool-path errors and avoidable assistant output have different inputs and output schemas. Task-specific runtime records are summarized after each static instruction.

C.1 CURRICULUM PLANNING AND TASK CONSTRUCTION

Prompt 1: Curriculum planner

SYSTEM (excerpts):

Allocate a reusable curriculum from the Student diagnosis and COMPLETE seed roster. Use `curriculum_allocation` once. Return only `purpose_weights` and compact lanes; do not author task goals, fields, values, verifier logic, solving steps or implementation plans.

For each seed group choose `runtime_seed_options`, `mutation_axes` and `change_strength`. The axes are control coordinates, NOT a task/goal/family whitelist. The local Builder receives the full original goal and implementation materials and chooses the concrete source-grounded business change.

USER:

Runtime input: *Student capability diagnosis, complete seed roster, available MCP catalogue, curriculum policy and delivery ledger.*

Figure 6: Excerpts from the curriculum-planner system prompt. The output assigns learning directions and seeds to construction work orders.

Prompt 2: Task builder

SYSTEM (excerpts):

Realize ONE curriculum work order as a source-grounded task specification. The Controller selected runtime seeds, one mutation axis and a change strength; it did NOT author a concrete goal. You own the local business change after reading the supplied original goal and `implementation_surface`. Seeds are initial data/environment plus an example goal, not a mandatory final goal.

Public tasks state the business outcome/rules, never tool sequences, Student weaknesses, grader internals or private answers. Private `semantic_delta` explains what actually changes.

Use `task_specification` with `action=build` and `task={operation,parameters}`. The framework computes references, writes files, packages the unchanged seed environment, and executes the same verifier entry used by the harness.

USER:

Runtime input: *Work order and learning target, full seed request, indexed source materials, operation APIs, sibling goals, and available MCP tools.*

Figure 7: Excerpts from the builder system prompt for requirement changes on selected seed environments. Recorded work orders and outputs appear in the cases in Section D.1.

C.2 TASK REALIZATION

Prompt 3: Task realization

SYSTEM:

You realize a typed tool-use task as one natural user request. Rewrite the whole request; never append or concatenate parent prompts. Preserve every target goal, dependency, constraint, and required verbatim clause. Do not mention internal graphs, verifiers, operators, benchmarks, or evolution terminology.

Return JSON only with `prompt`, `covered_node_ids`, and `intent_summary`.

USER:

Runtime input: *Typed task program, parent prompts, surface contract, and target signal.*

Figure 8: Static instruction used to realize a structured task as a user request.

C.3 FAILURE ATTRIBUTION

Prompt 4: Failure attribution

SYSTEM:

You are a control-plane component for an executable MCP self-evolution system. Return one JSON object only. Do not invent executed evidence. Separate model behavior from environment, harness, and verifier or task-contract failures.

USER:

Diagnose only the supplied failed teacher rollouts. Return `primary_attribution`, `failure_atoms`, `capability_contract`, `confidence`, `evidence`, `recommended_route`, `affected_attempt_ids`, `failure_type`, and `failure_subtype`. Attribute model behavior only when it is supported by the teacher trace and unchanged verifier.

Runtime input: *Public task, failed trace, verifier outcome, and visible execution evidence.*

Figure 9: Static instruction used for teacher-failure attribution.

C.4 RECOVERY-SKILL DISTILLATION

Prompt 5: Recovery-skill distillation

SYSTEM:

You are a control-plane component for an executable MCP self-evolution system. Return one JSON object only and do not invent executed evidence.

USER:

Compare the failed teacher trajectory, the agent-visible request, tool responses, final artifacts, and the failed verifier contract. Decide whether the failure is `recoverable` or a `task_contract_mismatch`. If recoverable, return one temporary repair skill with `skill_id`, `trigger`, `procedure`, `stop_condition`, `anti_patterns`, and `scope`. The procedure may improve planning, tool use, checking, or recovery, but must not change the task, environment, verifier, expected answer, or available tools. Ground every step in supplied evidence. Do not encode verifier-only criteria, hidden reference answers, or information unavailable to the teacher during execution. This is teacher-only generation context.

Runtime input: *Failed rollout, diagnosis, visible evidence, and compatible skills.*

Figure 10: Static instruction used to distill a source-grounded recovery skill.

C.5 TRAJECTORY-EFFICIENCY ASSESSMENT

Prompt 6(a): Tool-path efficiency audit

SYSTEM:

You are a strict trajectory-efficiency auditor. A redundant successful call is wholly avoidable: it repeats already-known evidence or never contributes to a required inference. Do not mark a call merely because another strategy was possible. A real tool error means that the call failed, timed out, was rejected, returned an explicit error, or used invalid arguments. Use only listed call IDs.

Return JSON with `redundant_successful_call_ids`, `additional_real_error_call_ids`, and `notes`.

USER:

Runtime input: *Public task and ordered tool-call summaries.*

Figure 11: Static instruction used for tool-path efficiency assessment.

Prompt 6(b): Assistant-output efficiency audit

SYSTEM:

For every assistant-output segment, estimate the fraction that is avoidable without reducing correctness, evidence traceability, or task compliance. Necessary planning, evidence interpretation, calculations, and final synthesis are not waste. Repetition, abandoned narration, needless restatement, and unrelated text are waste. Choose one value from $\{0, 0.05, 0.10, 0.20, 0.30, 0.40, 0.50, 0.75, 1.0\}$ and return each supplied segment identifier exactly once.

USER:

Runtime input: *Public task, tool-path summary, and ordered assistant segments.*

Figure 12: Static instruction used for output-efficiency assessment.

D QUALITATIVE CASES

The following figures show recorded requirement changes, environment variants, and a requirement change on an evolved environment. The task cases connect the changed request or state to the corresponding verification. The trajectory figure abridges a recorded recovery while preserving its tool calls, intermediate values, and verifier-grounded failure atoms.

D.1 TASK-EVOLUTION CASES

Case A: replacing a single-run goal with checkpoint-wise reconciliation

Feedback and planner assignment. The capability report identifies incomplete collection of experiment and research objects and incomplete structured deliverables. The recorded assignment selects the W&B seed with `purpose=explore`, `mutation_axis=goal_replacement`, and `change_strength=structural`.

Before: seed requirement

Source. A W&B project with five completed runs, each containing 500 logged steps.

Goal. Select the run with the smallest final response length. Record that run's entropy loss, clip ratio, and mean response length at checkpoint zero and subsequent 100-step checkpoints.

Output. `shortest_length_experiment.csv`.

Selection. One final winner supplies the metrics across checkpoints.

After: evolved requirement

Source. The same project and run histories.

Goal. At each checkpoint, compare all five runs and select the one with the smallest mean response length. Record its metrics and whether it is also the final-step winner. Include steps 0, 100, 200, 300, 400, and 499.

Output. `checkpoint_leader_reconciliation.csv`.

Selection. The leading run is determined separately at every checkpoint.

Builder and validation. The builder patches the seed evaluator for the new table and selection rule. It accepts two positive cases and rejects seven negative cases, including the old single-winner table and an omitted final checkpoint. A teacher rollout then passes the native verifier.

Figure 13: **Requirement evolution.** The planner supplies the learning direction; the builder chooses a concrete change supported by the seed data. The recorded assignment is exploratory. Both the output requirement and its verifier change, while the source environment is reused.

Case B: changing memory facts and the live form layout

Shared request. Complete the welcome-party form for two people using their stored memory. Both attend the full day; unstated information defaults to negative. If a person has multiple responses, the latest one counts.

Environment instance: s18

Memory facts.

Elena Brooks S0018: No Seafood.
Talia Nguyen S0018: None.

Form state. Eleven questions in a form created for this instance. `Name` is question 3; `Dietary Restrictions` is question 4.

Dietary option order. None; No Seafood; No Spicy Food; Kosher; Vegan.

Expected responses. Bound to the s18 people, facts, and form.

Environment instance: s21

Memory facts.

Noah Patel S0021: Vegan.
Priya Martin S0021: No Seafood.

Form state. Eleven questions in a separately initialized form. `Name` is question 7; `Dietary Restrictions` is question 11.

Dietary option order. No Spicy Food; None; No Seafood; Vegan; Kosher.

Expected responses. Bound to the s21 people, facts, and form.

Validation. The request and MCP tool interfaces are the same. The evaluator retains exact scalar matching, checkbox-set matching, and the latest-response rule. Local checks reject wrong choices, incomplete attendance, and stale responses. Teacher rollouts pass the native verifier in both instances.

Figure 14: **Environment variation.** Recorded instances from the paired environment evaluation change observable facts and layout while retaining the capability goal. Expected response values and the target form are rebound to each instance. The held-out s21 instance and its teacher trajectory are excluded from SFT.

Case C: requirement evolution on an evolved Canvas environment

Environment preparation. The seed varies course names, teacher assignments, meeting times, schedule order, and syllabus-file mappings. Its emailed schedule contains 27 assigned courses taught by 11 instructors. The goal change below operates on this initialized environment.

Feedback and planner assignment. The capability report identifies incomplete Canvas side effects and incomplete entity coverage. The planner selects `purpose=remediate`, `mutation_axis=grouping`, and `change_strength=structural`.

Before: per-course announcements

Example instructor. Rebecca Richardson teaches:

Modern Poetry — Cedar Term 68;
American Literature — Alder Term 68;
Developmental Psychology — Harbor Term 68.

Requirement. Publish an announcement in each assigned course stating its instructor and class time.

Other obligations. Create each course, enroll its instructor, upload its syllabus, and publish the course.

After: announcements grouped by instructor

Same schedule and course resources.

Requirement. For each instructor, publish the consolidated notice in the first course in schedule order. Name that instructor and every course in the group using the full scheduled titles. Other courses in the group must not carry a notice naming that instructor.

Example destination. Modern Poetry — Cedar Term 68 carries the notice for Rebecca Richardson's three courses.

Preserved. Course creation, teacher enrollment, syllabus upload, and course publication.

Builder and validation. The builder groups by instructor, chooses the first course, and updates the announcement checks. It accepts one positive case and rejects six negative cases, covering missing group members, a notice in the wrong course, and missing retained side effects. The teacher passes the native verifier.

Figure 15: **Combining environment and requirement changes.** Environment variation first supplies the executable seed; feedback-guided requirement evolution then changes announcement routing on that seed. The final task uses 11 instructor-group destinations while preserving the remaining requirements for all 27 courses.

D.2 TRAJECTORY-EVOLUTION CASE

Same task; skill-guided recovery proceeds from left to right.

Before: failed trajectory FAIL

1. Locate data `list-collections` → six schema inspections

2. Aggregate `group(Game Genre, count)`
Counts were collected without the required sales evidence.

3. Select samples `find(..., limit=1)`
The returned documents did not match the required samples.

4. Reconcile groups Top 1: **423** vs. remainder: **2,371**
The required top-four–tail comparison was not performed.

5. Final evidence Average Daily Sales and the tail's sales contribution were absent.

10 turns | **31** tool calls | claim coverage: **0.50**

After: recovered trajectory PASS

1. Locate data `list-collections` → four focused schema inspections

2. Aggregate `group(Game Genre, count, sales)`
Counts, average sales, and total sales were collected together.

3. Select samples `find(..., sort=_id)` + count checks
The deterministic samples agreed with the grouped counts.

4. Reconcile groups Top 4: **1,666** vs. tail: **1,128**
The two groups account for all **2,794** documents.

5. Final evidence Tail sales: **5,792.16 / 13,663.49** (**42.4%**).

7 turns | **20** tool calls | claim coverage: **1.00**

Figure 16: **Trajectory-evolution case.** The two columns show an abridged trace for the same task. Red rows identify verifier-grounded failures; blue rows show the corresponding skill-guided corrections. The tool calls, intermediate values, and footer statistics come from the complete recorded executions. Claim coverage is the score reported by the task verifier.

E Additional Experimental Analyses

E.1 Benchmark feedback and evaluation

The task sets from Toolathlon, MCPMark, and MCP-Atlas supply student rollout feedback and are also used to score the checkpoints in Tables 1 and 2. After each student update, rollouts and task outcomes on these benchmark tasks inform the learning targets and seed-task selection for the next evolution round. We evaluate successive checkpoints on the same task sets. The reported scores therefore measure improvement under benchmark-guided feedback across rounds.

E.2 Student progress on the feedback pool

We align the existing MCP-Atlas diagnostic records by task ID. The three checkpoints share 310 tasks with identical queries and reference claims. Their Pass@1 scores are shown in Table 6. The initial checkpoint is the student before the reported evolution rounds, rather than the seed-only SFT baseline.

Table 6: **Student performance on the common diagnostic tasks.**

Checkpoint	Tasks	Passed	Pass@1 (%)
Initial	310	196	63.23
EvoMCP R1	310	221	71.29
EvoMCP R2	310	228	73.55

Eight tasks lose Lara translation tools after the initial evaluation. For the paired analysis, we use the remaining 302 tasks, whose queries, reference claims, and enabled tool-name lists are unchanged. Table 7 groups tasks by the initial outcome, so that improvement on failures and retention of prior successes are both visible.

Table 7: **Resolving initial failures and retaining initial successes.** Percentages use the number of tasks in each initial group.

Initial outcome	Tasks	R1 passed	R2 passed
Failed	113	60 (53.10%)	65 (57.52%)
Passed	189	154 (81.48%)	155 (82.01%)
All	302	214 (70.86%)	220 (72.85%)

Relative to the initial student's 189/302 passing tasks, R2 gains 65 passes and loses 34, for a net increase of 31 tasks or 10.26 percentage points. These diagnostic outcomes describe changes in observed failures. The feedback ablation in Table 2 separately tests the effect of feedback-guided curriculum planning.

Variation across reference workflows. We also group the same 302 tasks using fixed reference trajectories. Calculation-related workflows contain a calculator, code executor, or MongoDB aggregation call. Service counts describe the services used in the reference execution. Table 8 shows that changes differ across groups. Calculation-related performance peaks at R1, while tasks whose references use at least three services continue to improve at R2.

Table 8: **Diagnostic Pass@1 by reference workflow.**

Reference workflow	N	Initial	R1	R2
Calculation-related	99	56.57	73.74	70.71
Other workflows	203	65.52	69.46	73.89
One service	5	100.00	100.00	80.00
Two services	209	64.11	75.12	76.08
At least three services	88	56.82	59.09	64.77

E.3 A BEHAVIOR-LEVEL CORRECTION

One diagnostic task requests apartments built after 2010, subject to additional housing constraints, and asks for the cheapest apartment and the mean rent. Table 9 traces a concrete correction in the student's tool arguments and answer.

Table 9: **Correcting a strict filter boundary.** All entries come from recorded tool calls and answers for the same task.

Observed behavior	**Initial**	**R1**	**R2**
YearBuilt predicate	≥ 2010	> 2010	> 2010
Selected apartments	12	11	11
Reported mean rent	3,293.583	3,387.182	3,387.182
Recorded claim coverage	0.167	1.000	1.000

The initial student includes apartments built in 2010. Both evolved checkpoints apply the strict boundary and compute the mean over the resulting 11 apartments. This case identifies a specific learning target: translating a verbal boundary into the correct tool predicate and carrying the selected set consistently into aggregation. R2 still reports an incorrect auxiliary sum in its final answer, despite the correct mean and a passing judge verdict.

E.4 TASK-EVOLUTION MEASUREMENTS

The following protocol specifies the paired comparison in Table 3(a).

Sample and round. The sample consists of 2,400 single-parent seed–evolved task pairs from task acquisition before the first student update, denoted R1. Eligible evolved tasks have complete lineage and pass task-validity checks. Pairs are sampled uniformly without replacement using random seed 42 before filtering on teacher success. Each evolved-task ID appears once, although a seed may support different targets or descendants. Multi-seed constructions are outside this paired comparison. The target is the planner's recorded pre-generation target, shared by both members and kept unchanged during scoring. The pair record retains task versions, the target, and its retain, remediate, or explore purpose. Each pair has equal weight.

Target realization. A task realizes its target when its requirements necessarily exercise the selected behavior, its observable environment provides the information or resources needed for that behavior, and its verifier checks the corresponding requirement. The judgment must identify evidence for all three conditions in the task instruction, environment resources, and verifier code or tests. A target label, a changed instruction, or a successful rollout alone is insufficient. The seed and evolved task receive separate judgments against the same target. Both can be positive when the seed already exercises that behavior.

Behavior change. An evolved task is positive when the comparison identifies a changed necessary action, dependency, aggregation rule, or decision conditioned on observable state. For an environment transformation, the evidence must explain how changed facts alter the required action, lookup, or tool arguments. Wording changes and incidental differences between sampled trajectories do not qualify. The comparison records the required behavior before and after evolution and cites the task evidence establishing the difference. Behavior change is reported only for evolved tasks, with seeds serving as the reference.

Judge and consistency check. GLM-5.2 is the fixed audit judge, with temperature 0 and an 8,192-token output limit. Its input contains the unchanged learning target and both task packages, including instructions, relevant environment resources, verifier checks, and verifier-test evidence. Generator rationales, student and teacher outcomes, and round or source labels are withheld. Two separate calls present the packages as A and B in opposite orders. Each call returns target-realization labels for A and B, a behavior-change label, and source locations supporting every label. Labels are positive, negative, or insufficient evidence. A positive score requires both calls to agree after

restoring task identities and to supply the evidence required above. All other cases score zero, with disagreements and insufficient-evidence cases retained in the audit records and denominator. These are agreement-confirmed model judgments, not human annotations.

> **Audit instruction.** Assess both task packages against the supplied learning target. For each package, identify the necessary task behavior, the accessible information or resources that support it, and the verifier check that tests it. Label target realization positive only when all three are supported. Then determine whether solving the two tasks requires a substantive difference in actions, dependencies, aggregation, or decisions based on observable state. Cite the before–after requirements and their source locations. Ignore wording-only changes, generation claims, and presumed model difficulty. Return a label and evidence for each judgment. Use insufficient evidence whenever a required link cannot be established.

Student and teacher execution. The student is the initial Qwen3.6-27B checkpoint, R0, on both sides. The teacher is Qwen3.8-Max, using the same endpoint revision for all paired runs. Each actor receives one rollout per task from a fresh initial state with the standard task-execution prompt and declared tools. The paired protocol uses the Toolathlon single-turn harness with a 200-step agent limit, a configured 50-turn outer limit, temperature 0, maximum reasoning effort where supported, and a 262,144-token completion ceiling per model request. These limits match across the seed and evolved sides. Teacher rollouts receive no recovery skills or refinement guidance. Each task's own verifier determines success. A completed failing rollout, incorrect tool use, or exhaustion of the execution budget counts as failure. Infrastructure faults trigger a fresh-state rerun under unchanged settings and do not count as model failures. Execution rates are computed only after all 2,400 pairs have determinate outcomes. Existing seed demonstrations do not substitute for these matched executions.

Aggregation and interpretation. All percentages in Table 3(a) use the full set of 2,400 pairs. Target realization counts confirmed-positive tasks on each side, and behavior change counts confirmed-positive pairs. Student failure and teacher success count the respective verifier outcomes for R0 and the teacher. The difference column reports evolved-minus-seed differences in percentage points. The student failure rate has no universally preferred direction: retain targets seek to preserve an established behavior, while remediate and explore targets can expose unmet demands. Target realization can remain unchanged when evolution varies conditions for behavior already required by the seed.

Trajectory measurements. The first-round (R1) comparison in Table 3(b) uses 2,400 paired teacher demonstrations before and after refinement. Each pair shares the same task and verifier, with verifier-passing demonstrations on both sides. For each condition, we average tool-call errors, AWR, and generated output tokens across demonstrations. Errors and AWR follow the trajectory audit.

Round-level measurements. The task-validity, trajectory-acceptance, and training-pair-admission rates in Figure 3(b) are computed from the actual evolution records in each round, using the observed task counts. Trajectory acceptance includes verifier-passing teacher demonstrations obtained through failure recovery. For Figure 3(c), turns and tool calls are measured from teacher trajectories, with each measure normalized by its median at R0.

Matched recovery comparison. Figure 3(d) compares unguided retries and skill-guided recovery on model-behavior failures eligible for retry in each round. The two conditions use the same eligible failed tasks and a matched teacher-rollout budget. The recovered-task share is the number of these failures that yield a verifier-passing demonstration within that budget, divided by the total number of tasks in the corresponding round.