WEFT: SCALING TOOL-USE POST-TRAINING FOR GENERAL-PURPOSE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent efforts to scale tool-use post-training have largely centered on the synthesis of executable environments, which constitute only one component of a broader agentic interaction system comprising the environment, task, agent harness, and evaluator. Scaling environments in isolation, however, does not guarantee commensurate gains in model performance, because reliable learning signals depend on coherent interactions among all components of the agentic interaction system. To address this problem, we introduce WEFT (Whole-system Evolution For Tool-use Post-training), which couples scalable agentic interaction system construction, execution-driven self-evolution, and stable post-training. WEFT scales agentic interaction system construction across environment breadth, task complexity, and interaction diversity. Execution-driven self-evolution iteratively uses execution traces and state evidence to attribute failures and revise the responsible components, with fresh rollouts evaluating the changes and providing evidence for subsequent evolution rounds. For stable post-training at scale, WEFT addresses both optimization and execution reliability: prefix-preserving sampling retains verified progress and atomic-turn credit assignment localizes learning signals, while MegaMCP maintains isolated, recoverable state across concurrent rollouts over shared tool services. Extensive experiments across various models and benchmarks demonstrate the effectiveness of WEFT for tool-use post-training. WEFT-8B and WEFT-14B outperform all evaluated matched-size environment-scaling baselines on BFCL V4, τ^2-Bench, and Claw-Eval. In particular, WEFT-14B improves over Agent-World-14B by 6.41, 2.23, and 12.27 percentage points. WEFT-35B-A3B further extends these gains to more challenging long-horizon workflow benchmarks, including Toolathlon-Verified and AutomationBench.

1 INTRODUCTION

Large language model (LLM) agents use tools to interact with stateful environments and accomplish complex, multi-step tasks. Solving these tasks requires agents to track environment state changes and adapt their actions to environment feedback (Zhou et al., 2024; Xie et al., 2024; Trivedi et al., 2024; Yao et al., 2025). Effective post-training of such agents depends on diverse interaction trajectories and reliable supervision based on task outcomes. Recent work has explored the programmatic synthesis of executable environments to expand the range of tools and task scenarios available for post-training (Song et al., 2026b; Dong et al., 2026; Wang et al., 2026; Tu et al., 2026).

However, scaling executable environments alone does not ensure reliable learning signals. Training experience emerges from an agentic interaction system comprising the environment, task, agent harness, and evaluator. An executable environment does not guarantee that tasks are feasible or their outcomes are correctly evaluated. Task failures may arise from faulty tool implementations, infeasible tasks, or verification errors, rather than model limitations alone. Attributing these failures to the policy can penalize valid behavior and introduce misleading learning signals (Cai et al., 2025a; Xu et al., 2026b). Effective scaling therefore requires not only broader environment coverage, but also continual improvement of the complete agentic interaction system. Execution experience must support both policy learning and the identification and revision of problematic components.

To this end, we introduce WEFT, an end-to-end tool-use post-training framework integrating scalable agentic interaction system construction, execution-driven self-evolution, and stable training. It com-

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.

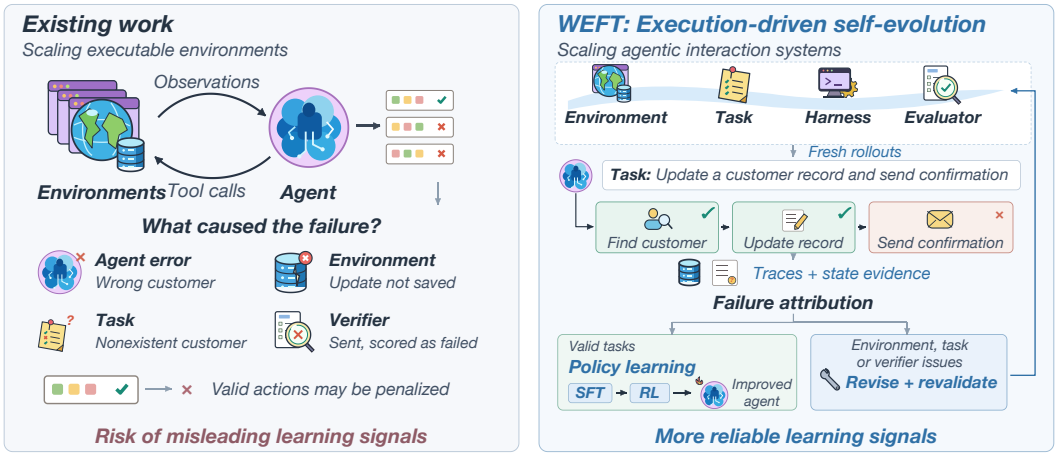


Figure 1: WEFT improves the agentic interaction system through execution feedback. Fresh rollouts evaluate these changes and guide the next round of refinement toward more reliable learning signals.

bines stateful environment synthesis and verifiable cross-tool task composition with diverse task-disclosure settings and agent harnesses to jointly scale environment breadth, task complexity, and interaction diversity. Using execution traces and state evidence, WEFT distinguishes policy failures on valid tasks from environment, task, or verifier issues, guiding targeted component revisions to improve the reliability of training experience and learning signals at their source. Fresh rollouts assess revisions and uncover new issues for continued evolution. For efficient, stable post-training, prefix-preserving rejection sampling constructs high-quality trajectories by retaining verified progress and resampling only failed atomic tasks. For reinforcement learning (RL), task filtering assesses verifier reliability to select training tasks, while atomic-turn credit assignment provides finer-grained learning signals. MegaMCP avoids repeated MCP deployment by sharing tool services across concurrent rollouts. It isolates each rollout's database and workspace state, using snapshots to support retries and candidate sampling from identical states.

We post-train Qwen3-8B, Qwen3-14B, and Qwen3.5-35B-A3B with WEFT and evaluate the resulting models on five benchmarks spanning tool calling, multi-turn interaction, and long-horizon task execution. WEFT-8B and WEFT-14B outperform all evaluated matched-size environment-scaling baselines on the aggregate scores of BFCL V4 (Patil et al., 2025), τ^2-Bench (Barres et al., 2025), and Claw-Eval (Ye et al., 2026). WEFT-35B-A3B achieves 45.99% on Toolathlon-Verified (Li et al., 2026) and 26.50% on AutomationBench (Shepard & Salimans, 2026). Controlled experiments further demonstrate the value of continually improving the agentic interaction system: with the task set and rollout budget fixed, three self-evolution rounds reduce the tool-call error rate in selected training trajectories by 45.5% relative to the initial construction and improve three downstream benchmark scores by 3.65–5.25 percentage points. Ablations also show performance gains from diverse task-disclosure views and agent harnesses, prefix-preserving rejection sampling, and atomic-turn credit assignment. Systems experiments show that MegaMCP reduces data transfer, memory use, and cold-start latency during concurrent rollouts.

Our main contributions are:

- We introduce WEFT, an end-to-end tool-use post-training framework that expands the scope of scaling from executable environments to complete agentic interaction systems.
- We develop execution-driven self-evolution of the agentic interaction system for reliable learning signals. Execution experience guides failure attribution and targeted component revisions. Fresh rollouts assess changes and uncover new issues for subsequent evolution rounds.
- We support stable post-training through algorithmic design and execution infrastructure. We develop efficient prefix-preserving rejection sampling to construct high-quality trajectories for supervised fine-tuning. In reinforcement learning, task filtering screens verifier reliability, and atomic-turn credit assignment provides finer-grained learning signals. For infrastructure, MegaMCP maintains isolated, recoverable state across concurrent rollouts over shared tool services.

2 Related Work

Environment and task scaling. EnvScaler, ScaleEnv, and AWM scale tool-use training through stateful environments and verifiable tasks (Song et al., 2026b; Tu et al., 2026; Wang et al., 2026). ClawGym combines workspace task synthesis, hybrid verification, and sandbox rollouts (Bai et al., 2026a). Agent-World combines environment-task discovery with multi-environment RL, using capability-gap diagnosis to generate targeted tasks (Dong et al., 2026). WEFT uses execution traces and state evidence to iteratively refine environments, tasks, and verifiers, testing revisions through fresh rollouts.

Agent interaction and rollout systems. APIGen-MT studies simulated agent-user interaction for multi-turn data generation, while HarnessBench and ClawGym examine harness and workspace effects on execution (Prabhakar et al., 2025; Yao et al., 2026; Song et al., 2026a). Agentic Harness Engineering uses execution traces to evolve harnesses, while Agent Lightning separates agent execution from training (Lin et al., 2026; Luo et al., 2025). WEFT presents the same grounded task through distinct interaction formats and native harnesses, while MegaMCP supports concurrent rollouts with shared tool services and isolated, recoverable state.

Agent post-training and self-improvement. Tool-use data pipelines combine execution or semantic checks with supervised training (Liu et al., 2024; 2025; Fu et al., 2025). Other work improves reinforcement learning through state-aware grouping or local branching for finer-grained credit assignment (Feng et al., 2025; Yi et al., 2026; He et al., 2026). WEFT retains verified prefixes for SFT and assigns credit at atomic-task boundaries for RL, coupling policy learning with execution-driven refinement of the agentic interaction system. Appendix E provides the full discussion.

3 Method

WEFT organizes tool-use post-training around the agentic interaction system, with execution supporting both policy learning and continual system improvement. We jointly expand environment breadth, task complexity, and interaction diversity through stateful environment synthesis, verifiable task composition, and rollouts across multiple task views and agent harnesses (Section 3.1). Execution traces and state evidence support outcome verification, failure attribution, and targeted component revision; fresh rollouts with the updated system assess the changes and inform subsequent revisions (Section 3.2; Figure 2). To support stable post-training on these interactions, we combine prefix-preserving rejection sampling, task filtering, and atomic-turn credit assignment with MegaMCP's isolated, recoverable concurrent execution (Section 3.3).

3.1 Constructing Scalable Agentic Interaction Systems

Scaling agentic interaction systems requires not only diverse tool environments, but also coherent tasks and varied interaction settings. WEFT synthesizes stateful environments from tool specifications, generates and validates reusable atomic tasks, and composes them into cross-MCP workflows with explicit dependencies and grounded initial states. We vary task-disclosure settings and agent harnesses while preserving task objectives and completion conditions. Together, these choices expand environment breadth, task complexity, and interaction diversity. The resulting rollouts provide execution traces and state evidence for subsequent verification and self-evolution.

3.1.1 Stateful Environment Synthesis

Shared state and execution contracts. We construct MCP tool definitions from public tool documentation and business requirements, specifying descriptions and input and output schemas. We define two types of state for tool operations, *database state* and *workspace state*; tools may operate on either or both. An environment synthesis agent infers shared entities, read and write dependencies, and state transitions from the complete tool set. It instantiates a shared database schema to validate types and relational constraints. The schema and MCP definitions guide contracts specifying argument-to-state mappings, reads, writes, and returns; these artifacts jointly guide FastAPI backend generation. For workspace state, existing file-oriented runtimes bind to isolated workspaces.

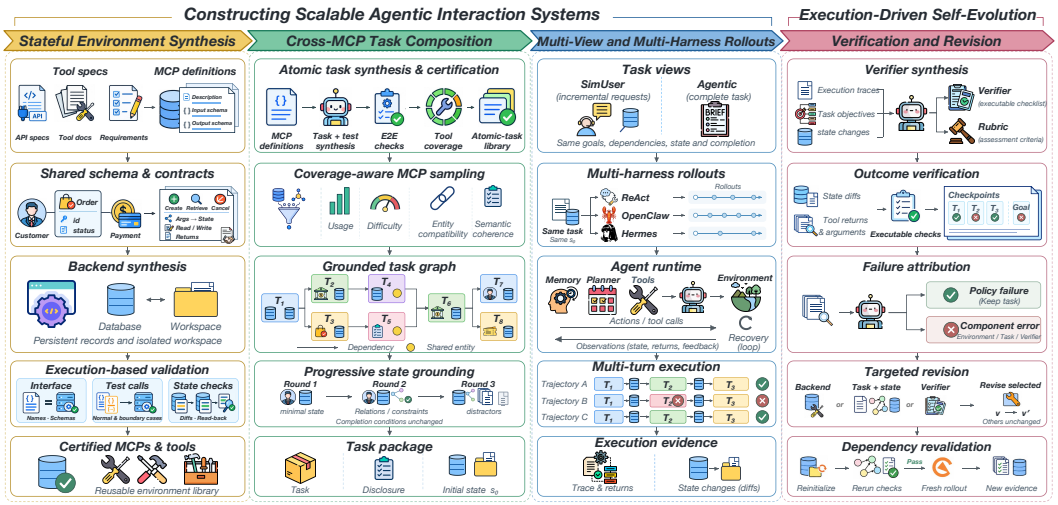


Figure 2: Construction and execution-driven self-evolution in WEFT. Certified atomic tasks form grounded cross-MCP workflows with resettable state. Views and harnesses diversify rollouts, whose traces and state evidence guide verification, failure attribution, and iterative component revision.

Execution-based validation. Running interfaces are checked against MCP tool names and argument schemas, then tested with generated normal and boundary calls. Queries are checked against state; updates against database or workspace differences and subsequent reads. Structural, mapping, and runtime or state-access errors trigger schema, contract, and backend revisions, respectively. Backends passing revalidation become independently executable MCP servers.

3.1.2 CROSS-MCP TASK COMPOSITION

Certified atomic tasks. A task synthesis agent uses each certified MCP's definitions, schema, and contracts to generate *atomic tasks* and end-to-end (E2E) tests. An atomic task is an MCP's smallest independently executable and verifiable unit, realizing one intent through bounded tool calls. Construction uses *(i) E2E execution validation*, checking completion against returns and persistent state after real calls from independent initial states; *(ii) tool-chain orthogonality*, pruning identical or contained ordered chains; *(iii) tool coverage*, measuring the fraction of MCP tools covered by retained chains. Failures, redundancy, and uncovered tools guide incremental generation and revision of the reusable task and test library, preserving validated tasks (Appendix A.1).

Coverage-aware sampling. Configurable profiles specify ranges for atomic-task count, MCP count, and domain breadth; corpus proportions set generation quotas. Sampling favors underrepresented domains and MCPs using capacity and usage statistics, reserving surplus capacity for composition. Candidates are ranked by shared entities, complementary operations, and functional redundancy. An evaluation model checks MCP descriptions for a coherent business objective. Usage is updated before sampling tasks from accepted combinations; rejections trigger resampling within the profile (Appendix A.2).

Grounded task graph. A composition agent links selected atomic tasks into a directed acyclic graph for a shared business objective: nodes specify intents, inputs, and outcomes; edges encode information and state dependencies. Entity bindings align business objects across MCPs. The graph distinguishes given information from information retrieved through tools. Deterministic checks ensure nodes use selected tasks, bindings are conflict-free, and inputs come from initial state or upstream outcomes. A stable topological order governs disclosure and execution.

Progressive state grounding. From the graph and MCP schemas, contracts, or workspace structures, a state-initialization agent builds initial state in three rounds: *(i) entity initialization* establishes target entities and minimal executable state; *(ii) relation completion* adds cross-tool relations and constraints; *(iii) context enrichment* adds historical records and distractors. All rounds preserve entity correspondence and completion conditions. States passing schema and task-reachability checks become resettable initial snapshots.

3.1.3 MULTI-VIEW AND MULTI-HARNESS ROLLOUTS

Task views. We vary request and information disclosure while keeping task objectives, dependencies, initial state, completion conditions, and MCP interfaces fixed across views and harnesses. Both views retain user-known inputs and upstream information. *(i) Agentic mode* provides a complete brief for autonomous planning. *(ii) SimUser mode* uses a user-simulation agent with persona sampling to disclose requests in dependency order. Each atomic task defines an *atomic-task turn* allowing multiple tool calls. A final reply without tool calls ends the turn; SimUser then issues the next request. Independently, *vague*, *partial*, and *full* vary disclosure explicitness.

Harness scaling. Our infrastructure routes the same task to ReAct, OpenClaw, or Hermes for sandboxed rollouts, retaining each harness's prompts, memory, planning loop, tool-call protocol, and error recovery. ReAct supports both views; OpenClaw and Hermes use Agentic mode.

3.2 EXECUTION-DRIVEN SELF-EVOLUTION

Rollouts across task views and agent harnesses produce training trajectories and can also reveal environment, task, or verifier problems missed by construction checks. WEFT uses execution traces and state evidence to verify task outcomes, distinguish policy errors from component problems, and revise the affected components. Fresh rollouts with the updated agentic interaction system evaluate these revisions and inform the next round, improving the reliability of learning signals through repeated execution and revision.

3.2.1 VERIFICATION FROM EXECUTION EVIDENCE

Verification checks atomic-task completion using *(i) state changes*, captured by database or workspace differences; *(ii) tool returns*. Given completion conditions and a reference execution accepted by a semantic evaluation model, a verifier-generating agent produces and saves deterministic checks and a corresponding rubric. The checks are tested on reference evidence and revised using execution feedback; subsequent rollouts need not follow the reference tool-call sequence.

For trajectory τ, K checkpoints cover atomic tasks and the overall objective. Each $r_k(\tau)$ is 1 when all checkpoint conditions hold and 0 otherwise; the trajectory score is their mean:

$$R(\tau) = \frac{1}{K} \sum_{k=1}^{K} r_k(\tau). \tag{1}$$

3.2.2 ATTRIBUTION-GUIDED REVISION

Failure attribution. An *attribution agent* examines task graphs, traces, checkpoint outcomes, and state evidence to distinguish policy failures on valid tasks from environment, task, or verifier problems. Patterns across views and harnesses help localize failures; disagreements between rubric-based and executable scores guide verifier revision. Policy failures alone leave tasks unchanged.

Targeted component revision. We route environment problems to backends, task or state problems to task graphs and initial states, and evaluation problems to verifiers. An *evolution agent* uses the current version and supporting evidence to revise only the selected components. Task semantics remain fixed across views within a version.

Dependency revalidation and iterative refinement. We re-instantiate affected environments and task states and rerun construction checks. Fresh rollouts on passing versions assess repairs and reveal unresolved or new problems. Alternating execution and revision drives iterative evolution of the agentic interaction system through its own execution experience.

3.3 STABLE TOOL-USE POST-TRAINING

Construction and self-evolution provide the environments and tasks for training, while stable post-training also requires high-quality training data and reliable learning signals. WEFT uses prefix-preserving rejection sampling to construct supervised fine-tuning data and combines task filtering with fine-grained rewards to provide reliable, stable learning signals for reinforcement learning (Figure 3). MegaMCP provides scalable rollout support for both training stages.

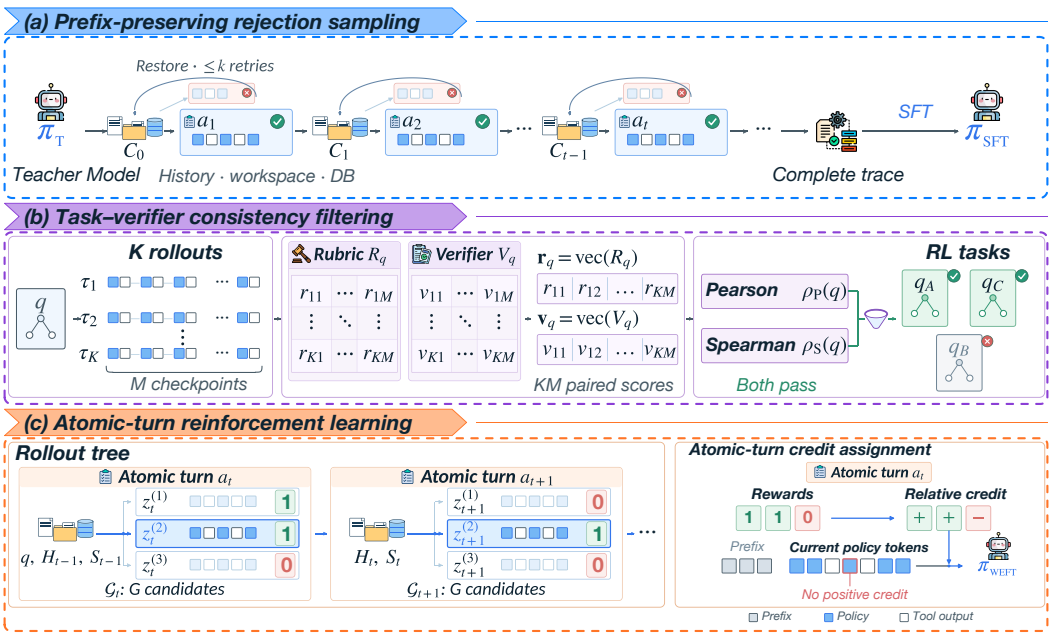


Figure 3: Post-training in WEFT. (a) Verified prefixes form SFT trajectories. (b) RL task filtering compares rubric-based and executable scores. (c) Atomic-turn RL assigns local credit, masking positive credit for detected undesirable patterns.

3.3.1 PREFIX-PRESERVING REJECTION SAMPLING

Rejecting complete trajectories after local failures discards successful progress. We use atomic-task turns as rejection units: the teacher executes tasks in dependency order, each checked by its verifier. Successful traces and database and workspace states are retained for subsequent tasks. On failure, we discard the attempt, restore pre-turn history and state, and resample only that task, with up to k additional retries. Once all tasks pass, retained segments form a complete trajectory for SFT.

3.3.2 REINFORCEMENT LEARNING WITH ATOMIC-TURN CREDIT ASSIGNMENT

Task and verifier consistency filtering. Verifiers may reject valid solutions or reward incomplete executions. Before RL, a rubric-guided LLM judge independently scores each task's pilot rollouts. We retain tasks only when both Pearson and Spearman correlations with executable scores meet selection criteria. RL uses only executable rewards.

Atomic-turn group-relative credit assignment. Rollouts follow a fixed topological order of the task graph, with one atomic task per turn. At turn t, a frozen rollout policy independently samples G segments from the same history in isolated, identical states restored by MegaMCP. Each segment may contain multiple tool calls. The atomic-task verifier gives $r_t^{(i)} = 1$ on completion and 0 otherwise. A uniformly selected passing segment (or failed segment if all fail) supplies history and state for the next turn. All candidates contribute to training.

We normalize rewards within each atomic-task group:

$$A_t^{(i)} = \frac{r_t^{(i)} - \bar{r}_t}{\mathrm{sd}(\mathbf{r}_t) + \epsilon}, \qquad \bar{r}_t = \frac{1}{G}\sum_{i=1}^{G} r_t^{(i)}, \tag{2}$$

where $\mathbf{r}_t$ contains group rewards and $\epsilon > 0$ stabilizes normalization. Clipped GRPO applies these advantages only to current-turn policy tokens, excluding prefixes, with token-mean aggregation across groups. Equal rewards give zero advantages, leaving only KL and entropy regularization.

Deterministic checks flag no-progress repetition, malformed tool-call or reasoning structures, and interface violations, even in successful segments. Flagged assistant turns receive token advantages $\min(A_t^{(i)}, 0)$, removing only positive credit. Unflagged turns, including later recovery, retain their advantages; KL and entropy regularization remain unchanged.

Table 1: **Main results on agentic tool-use benchmarks.** Scores (%) are grouped by total model parameters. Bold and underlined values mark the best and second-best reported scores in each group; dashes denote unreported results.

Method	BFCL V4								τ^2-Bench				Claw-Eval
	WebSearch	Memory	Multi-T.	No live	Live	Relev.	Irrelev.	**Avg.**	Retail	Telecom	Airline	**Avg.**	**Avg.**
Small- and Medium-Scale Open-Source Models (7B–32B)													
TOUCAN-7B	21.0	18.5	17.8	81.0	73.9	81.3	78.6	36.6	22.8	10.5	20.0	17.7	–
Qwen3-8B	7.0	17.6	35.4	90.2	80.9	81.3	77.2	40.4	34.0	18.0	26.5	26.2	25.6
Simulator-8B	17.5	6.0	4.1	47.6	44.6	31.3	**87.3**	23.9	32.2	29.2	34.0	31.8	–
EnvScaler-8B	23.0	21.9	47.1	88.5	82.2	**93.8**	74.6	47.6	49.6	32.7	31.5	37.9	22.6
AWM-8B	9.5	15.7	34.9	90.2	80.5	**93.8**	73.9	40.0	41.23	23.47	38.50	34.4	22.6
ScaleEnv-8B	–	–	–	–	–	–	–	–	50.9	27.2	37.5	38.5	–
EnvACE-8B	12.25	24.03	45.29	87.59	81.20	–	83.19	46.04	48.9	17.3	44.0	36.7	–
Agent-World-8B	47.0	21.7	44.5	83.3	79.6	**93.8**	80.2	51.4	72.8	50.9	40.0	61.8	30.5
Agent-World-14B	53.0	23.9	**53.9**	82.3	79.3	**93.8**	81.0	55.8	74.5	56.1	**52.0**	65.4	31.5
Qwen3-14B	4.0	19.8	36.9	90.0	**82.4**	81.3	79.4	41.0	55.3	14.9	27.0	32.4	24.7
AWM-14B	10.0	19.8	37.6	90.2	81.5	75.0	79.4	42.4	63.6	17.8	31.5	39.0	26.1
Qwen3-32B	26.0	15.7	43.3	**90.3**	82.0	81.3	82.4	46.7	59.5	27.2	48.0	44.9	–
WEFT-8B	49.5	33.12	47.75	77.9	69.5	81.25	66.88	52.28	71.93	72.81	42.0	62.25	35.78
WEFT-14B	**68.00**	**49.25**	52.38	79.02	66.69	62.5	84.75	**62.21**	**74.56**	**76.32**	**52.0**	**67.63**	**43.77**
Large-Scale Open-Source Models (35B–685B)													
nex-n2-mini-35b	**72.50**	39.14	**62.12**	72.02	61.44	56.25	**96.30**	63.94	74.56	73.68	**68.00**	72.08	**69.75**
Qwen3.5-35B-A3B	–	–	–	–	–	–	–	**67.3**	–	–	–	**81.2**	65.4
Qwen3-235B-A22B	54.0	23.9	45.4	37.4	68.9	**87.5**	81.7	47.9	71.9	58.0	45.6	58.5	–
DeepSeek-V3.2-685B	69.5	54.2	37.4	34.9	53.7	37.5	93.2	54.1	**81.1**	**96.2**	63.8	80.3	–
WEFT-35B-A3B	55.0	**66.02**	54.62	**73.5**	**71.21**	68.75	83.39	63.4	71.93	85.09	66.0	74.34	64.0

3.3.3 MEGAMCP: SHARED SERVING FOR ISOLATED ROLLOUTS

Shared services and private state. Prefix-preserving sampling and atomic-turn RL require repeated, concurrent task execution. MegaMCP shares MCP service processes across compatible sessions while task-scoped session groups manage each rollout's private databases and workspace, avoiding per-rollout MCP deployment.

State recovery and verification. Database and workspace snapshots support local retries for SFT and RL candidate sampling from identical states. Tool-call records and state evidence support task verification and reward computation; verifier code and privileged state remain inaccessible to the agent. Appendix D provides runtime and deployment details.

4 EXPERIMENTS

We evaluate downstream performance, isolate the contributions of training diversity, self-evolution, and post-training algorithms, and measure MegaMCP's overhead at rollout scale.

4.1 EXPERIMENTAL SETUP

Models and training data. We post-train Qwen3-8B, Qwen3-14B, and Qwen3.5-35B-A3B from their base models. We use Kimi K3 (Kimi Team, 2026) to generate supervised fine-tuning (SFT) trajectories for all 11,884 certified composed tasks (Appendix A.3). For reinforcement learning (RL), we select 1,954 of 5,000 candidate tasks using task and verifier consistency filtering (Section 3.3.2).

Evaluation. The main comparison and RL study use BFCL V4 (Patil et al., 2025), τ^2-Bench (Barres et al., 2025), and Claw-Eval (Ye et al., 2026). The data-construction ablations use Toolathlon-Verified (Li et al., 2026), AutomationBench (Shepard & Salimans, 2026), and Claw-Eval. Evaluation follows each benchmark's native harness and official aggregate metric.

4.2 MAIN RESULTS

WEFT-8B and WEFT-14B outperform all evaluated matched-size environment-scaling baselines on BFCL V4, τ^2-Bench, and Claw-Eval (Table 1). WEFT-14B achieves the highest aggregate scores in the 7B–32B group, exceeding Agent-World-14B by 6.41, 2.23, and 12.27 percentage points, respectively. WEFT-8B also improves all three scores over Agent-World-8B, by 0.88, 0.45, and 5.28

points. Claw-Eval shows the largest matched-size gain at both scales. The margins on BFCL V4 and τ^2-Bench are smaller for 8B and widen for 14B, which leads the 7B–32B group across all three aggregate metrics rather than on a single benchmark.

The category breakdown reveals recurring strengths in BFCL Memory and τ^2-Bench Telecom. Relative to the corresponding Agent-World models, WEFT-8B gains 11.42 and 21.91 points on these two categories, while WEFT-14B gains 25.35 and 20.22 points. WEFT-14B also improves BFCL WebSearch by 15.00 points, reaching 68.00%. Thus, the aggregate improvement is accompanied by gains across multiple categories, with particularly large margins on Memory and Telecom. These strengths persist in the larger model: WEFT-35B-A3B scores 66.02% on Memory, the highest reported score in the 35B–685B group and 11.82 points above the next-best result. Its Telecom score of 85.09% ranks second behind DeepSeek-V3.2 (96.2%) and exceeds nex-n2-mini-35b by 11.41 points. On the advanced benchmarks Toolathlon-Verified and AutomationBench, WEFT-35B-A3B achieves 45.99% and 26.50%, respectively (Table 2), extending the evaluation beyond the main benchmarks to more demanding multi-tool workflows.

4.3 UNDERSTANDING THE POST-TRAINING GAINS

Task Views and Harness Scaling With a fixed task set, we vary views under ReAct, then harnesses with the view mixture fixed (Table 2). Combining Agentic and SimUser outperforms either view on all benchmarks, averaging 3.18 points over Agentic. OpenClaw and Hermes yield a further 9.71-point mean gain. Both gains hold under benchmark-native harnesses.

Table 2: **Task views and harness scaling.** Benchmark scores (%) of WEFT-35B-A3B under different training-data configurations.

	Disclosure views			Harness scaling (cumulative)		
Benchmark	Agentic	SimUser	Both	ReAct	+ OpenClaw	+ Hermes
Toolathlon-Verified	27.78	19.44	**32.41**	32.41	39.51	**45.99**
AutomationBench	10.50	4.67	**13.67**	13.67	22.00	**26.50**
Claw-Eval	59.55	57.51	**61.29**	61.29	63.58	**64.00**

Execution-Driven Self-Evolution With the task set and rollout budget fixed, we compare the initial construction ($r = 0$) with up to three self-evolution rounds (Figure 4). Tool-error rate measures the fraction of tool calls returning errors in selected teacher trajectories, including errors from which the teacher recovers. Three rounds reduce tool-error rate from 1.76% to 0.96% (45.5% relative) and improve Toolathlon-Verified, AutomationBench, and Claw-Eval by 5.25, 4.33, and 3.65 points. Self-evolution improves both retained trajectories and downstream performance. The first two rounds capture most of the gains on AutomationBench and Claw-Eval, while Toolathlon-Verified continues to improve in the third round as tool errors decline monotonically.

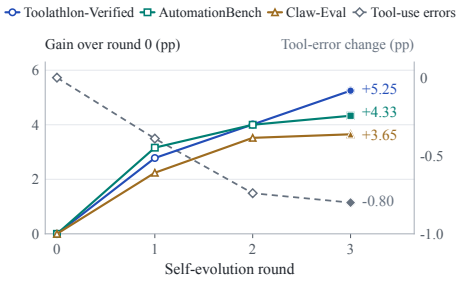


Figure 4: **Self-evolution across rounds.**

Prefix-Preserving Rejection Sampling Figure 5 compares prefix-preserving and end-to-end sampling with the task pool, verifier, and accepted-trace count fixed. Each allows $k \in \{1, 2, 3\}$ additional retries per atomic-task turn or full trajectory, respectively. At each retry limit k, prefix-preserving sampling outperforms end-to-end sampling. At $k = 3$, it gains 1.85 points on Toolathlon-Verified and 2.17 on AutomationBench, with Claw-Eval nearly tied.

Reinforcement Learning and Credit Assignment Atomic-turn GRPO improves on SFT across all three benchmarks at both model sizes (Table 3), with the largest gains on Claw-Eval: 4.59 points for 8B and 8.54 for 14B. The comparison with trajectory-level GRPO shows that these gains also depend on credit-assignment granularity. Trajectory-level GRPO improves BFCL V4 and Claw-Eval, with modest decreases of 3.17 and 0.84 percentage points on τ^2-Bench relative to SFT for 8B

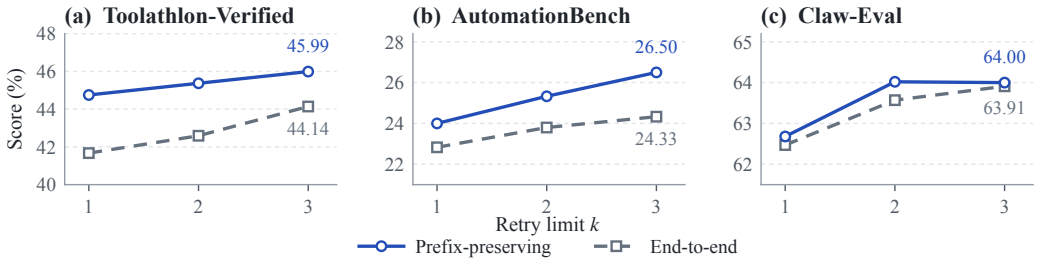


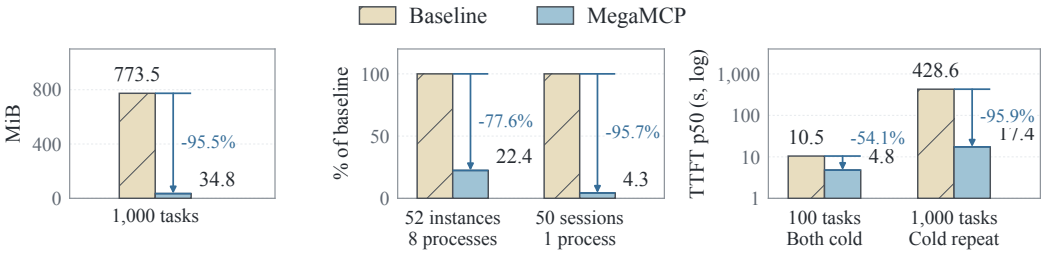
Figure 5: **Rejection sampling across retry limits.** Downstream scores of WEFT-35B-A3B.

and 14B, respectively. Atomic-turn GRPO instead improves all three benchmarks and exceeds the trajectory-level variant by 5.69 and 4.43 points on τ^2-Bench for 8B and 14B, respectively. These results support assigning credit at the atomic-task level rather than applying a single trajectory-level advantage to every turn.

Table 3: **RL gains and credit assignment.** Benchmark scores (%) after SFT and RL.

Training configuration	WEFT-8B			WEFT-14B		
	BFCL V4	τ^2-Bench	Claw-Eval	BFCL V4	τ^2-Bench	Claw-Eval
SFT	52.02	59.73	31.19	59.69	64.04	35.23
+ RL (trajectory-level GRPO)	52.25	56.56	34.92	60.47	63.20	41.64
+ RL (atomic-turn GRPO)	**52.28**	**62.25**	**35.78**	**62.21**	**67.63**	**43.77**

MegaMCP at Rollout Scale MegaMCP reduces three sources of rollout overhead (Figure 6). *(a) Data transfer.* Sharing tool services reduces total sandbox upload, including runner files, from 773.5 to 34.8 MiB at 1,000 tasks (95.5%). *(b) Memory.* Shared processes reduce resident memory by 77.6% for 52 instances in 8 processes and by 95.7% for 50 sessions in one process, relative to one process per session. *(c) Startup.* With both systems cold at 100 tasks, median time to the first tool call falls from 10.455 to 4.794 s (54.1%). Appendix D.2 details timing conditions and historical 1,000-task cold-repeat records.



(a) Upload volume **(b) Resident memory** **(c) Time to first tool call**

Figure 6: **MegaMCP resource use and startup latency.** (a) Total sandbox upload. (b) Memory relative to one process per session. (c) Median time to first tool call (log scale); 1,000-task bars show historical cold repeats. Local serving is the baseline in (a,c); conditions are in Appendix D.2.

5 CONCLUSION

WEFT shows that scaling tool-use post-training is fundamentally a systems problem rather than a matter of simply increasing the number of environments. Its central principle is to give execution experience a dual role: as data for policy learning and as evidence for improving the interaction system that produces it. Controlled experiments show that diversity across tasks, disclosure views, and agent harnesses broadens the training distribution; execution-driven self-evolution reduces tool-call errors and improves downstream performance; and stable sampling, fine-grained credit assignment, and scalable rollout infrastructure make these gains effective at scale. Overall, these findings support system-level scaling for reliable tool-use agents.

AI USE STATEMENT

In this work, we used generative AI tools for text polishing, including grammar correction and improvements to clarity, fluency, and presentation. We also used language models to generate synthetic training data, as described in Section 3 and the appendices. All AI-assisted edits were reviewed by the authors for correctness and faithfulness to the original technical content. We take responsibility for the final content of this work, including AI-assisted text, claims, and artifacts.

REFERENCES

Mohamed Salim Aissi, Clément Romac, Thomas Carta, Sylvain Lamprier, Pierre-Yves Oudeyer, Olivier Sigaud, Laure Soulier, and Nicolas Thome. Reinforcement learning for aligning large language models agents with interactive environments: Quantifying and mitigating prompt overfitting. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 7045–7061, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-naacl.390. URL https://aclanthology.org/2025.findings-naacl.390/.

Fei Bai, Huatong Song, Shuang Sun, Daixuan Cheng, Yike Yang, Chuan Hao, Renyuan Li, Feng Chang, Yuan Wei, Ran Tao, Bryan Dai, Jian Yang, Wayne Xin Zhao, and Ji-Rong Wen. ClawGym: A scalable framework for building effective claw agents, 2026a. URL https://arxiv.org/abs/2604.26904.

Hao Bai, Alexey Taymanov, Tong Zhang, Aviral Kumar, and Spencer Whitehead. Webgym: Scaling training environments for visual web agents with realistic tasks, 2026b. URL https://arxiv.org/abs/2601.02439.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Shihao Cai, Runnan Fang, Jialong Wu, Baixuan Li, Xinyu Wang, Yong Jiang, Liangcai Su, Liwen Zhang, Wenbiao Yin, Zhen Zhang, Fuli Feng, Pengjun Xie, and Xiaobin Wang. AutoForge: Automated environment synthesis for agentic reinforcement learning, 2025a. URL https://arxiv.org/abs/2512.22857.

Xin-Qiang Cai, Wei Wang, Feng Liu, Tongliang Liu, Gang Niu, and Masashi Sugiyama. Reinforcement learning with verifiable yet noisy rewards under imperfect verifiers, 2025b. URL https://arxiv.org/abs/2510.00915.

Aili Chen, Chi Zhang, Junteng Liu, Jiangjie Chen, Chengyu Du, Yunji Li, Ming Zhong, Qin Wang, Zhengmao Zhu, Jiayuan Song, Ke Ji, Junxian He, Pengyu Zhao, and Yanghua Xiao. DIVE: Scaling diversity in agentic task synthesis for generalizable tool use, 2026a. URL https://arxiv.org/abs/2603.11076.

Jiaao Chen, Jingyuan Qi, Mingye Gao, Wei-Chen Wang, Hanrui Wang, and Di Jin. Eigendata: A self-evolving multi-agent platform for function-calling data synthesis, auditing, and repair, 2026b. URL https://arxiv.org/abs/2603.05553.

Thibault Le Sellier de Chezelles, Maxime Gasse, Alexandre Lacoste, Massimo Caccia, Alexandre Drouin, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Graham Neubig, Quentin Cappart, Russ Salakhutdinov, and Nicolas Chapados. The BrowserGym Ecosystem for Web Agent Research. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=5298fKGmv3.

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. AgentWorld: Scaling real-world environment synthesis for evolving general agent intelligence, 2026. URL https://arxiv.org/abs/2604.18292.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training. In *Advances in Neural Information Processing Systems*, volume 38, pp. 46375–46408. Curran Associates, Inc., 2025. doi: 10.52202/085713-1544. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/420c9f777c0b4f78d515e53cf74d58b2-Abstract-Conference.html.

Dayuan Fu, Keqing He, Yejie Wang, Wentao Hong, Zhuoma GongQue, Weihao Zeng, Wei Wang, Jingang Wang, Xunliang Cai, and Weiran Xu. AgentRefine: Enhancing agent generalization through refinement tuning. In *International Conference on Learning Representations*, pp. 65185–65204, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a3cc50126338b175e56bb3cad134db0b-Abstract-Conference.html.

Kanishk Gandhi, Shivam Garg, Noah D. Goodman, and Dimitris Papailiopoulos. Endless terminals: Scaling rl environments for terminal agents, 2026. URL https://arxiv.org/abs/2601.16443.

Bowei He, Yankai Chen, Xiaokun Zhang, and Xue Liu. Branching policy optimization: Sandbox-native language agent reinforcement learning, 2026. URL https://arxiv.org/abs/2607.14171.

Mengkang Hu, Pu Zhao, Can Xu, Qingfeng Sun, Jian-Guang Lou, Qingwei Lin, Ping Luo, and Saravan Rajmohan. AgentGen: Enhancing Planning Abilities for Large Language Model based Agent via Environment and Task Generation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1*, pp. 496–507. ACM, 2025a. doi: 10.1145/3690624.3709321. URL https://doi.org/10.1145/3690624.3709321.

Shengran Hu, Cong Lu, and Jeff Clune. Automated Design of Agentic Systems. In *International Conference on Learning Representations*, pp. 21344–21377, 2025b. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/36b7acf6f6010652b3f2a433774a66fe-Paper-Conference.pdf.

Naman Jain, Jaskirat Singh, Manish Shetty, Tianjun Zhang, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural Environment Generation and Hybrid Verifiers for Scaling Open-Weights SWE Agents. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=7evvwwdo3z.

Kimi Team. Kimi K3: Open frontier intelligence, 2026. URL https://arxiv.org/abs/2607.24653.

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. In *International Conference on Learning Representations*, pp. 53359–53394, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/5783212d85c205ef823b8974d44872c5-Paper-Conference.pdf.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses, 2026. URL https://arxiv.org/abs/2604.25850.

Bo Liu, Simon Yu, Yiding Jiang, Ao Qu, Andrew Zhao, Zichen Liu, Junsu Kim, Zijian Zhou, Seungone Kim, Tongzheng Ren, Mickel Liu, Hanfei Yu, Zhaorun Chen, Weiyan Shi, Paul Pu Liang, Luke Zettlemoyer, Yejin Choi, and Natasha Jaques. SPADE: Self-play in adaptive synthetic executable environments, 2026. URL https://arxiv.org/abs/2608.19197.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. ToolACE: Winning the Points of LLM Function Calling. In *International Conference on Learning Representations*, pp.

41359–41381, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/663865ea167425c6c562cb0b6bcf76c7-Paper-Conference.pdf.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. APIGen: Automated PIpeline for Generating Verifiable and Diverse Function-Calling Datasets. In *Advances in Neural Information Processing Systems*, volume 37, pp. 54463–54482. Curran Associates, Inc., 2024. doi: 10.52202/079017-1725. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/61cce86d180b1184949e58939c4f983d-Paper-Datasets_and_Benchmarks_Track.pdf.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. 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. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-naacl.65. URL https://aclanthology.org/2025.findings-naacl.65/.

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K. Qiu, and Yuqing Yang. Agent Lightning: Train ANY AI agents with reinforcement learning, 2025. URL https://arxiv.org/abs/2508.03680.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large Language Model Connected with Massive APIs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 126544–126565. Curran Associates, Inc., 2024. doi: 10.52202/079017-4020. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/e4c61f578ff07830f5c37378dd3ecb0d-Paper-Conference.pdf.

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 *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 48371–48392. PMLR, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Manoj Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, Shelby Heinecke, Weiran Yao, Huan Wang, Silvio Savarese, and Caiming Xiong. APIGen-MT: Agentic Pipeline for Multi-Turn Data Generation via Simulated Agent-Human Interplay. In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025. doi: 10.52202/085713-2184. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/5e3661f7fe4c8ac5652d62eb3d3c96ea-Paper-Datasets_and_Benchmarks_Track.pdf.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *International Conference on Learning Representations*, pp. 9695–9717, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/28e50ee5b72e90b50e7196fde8ea260e-Paper-Conference.pdf.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Daniel Shepard and Robin Salimans. AutomationBench, 2026. URL https://arxiv.org/abs/2604.18934.

Dingfeng Shi, Jingyi Cao, Qianben Chen, Weichen Sun, Weizhen Li, Hongxuan Lu, Fangchen Dong, Tianrui Qin, King Zhu, Minghao Liu, Yuchen Eleanor Jiang, Jian Yang, Ge Zhang, Jiaheng Liu, Changwang Zhang, Jun Wang, and Wangchunshu Zhou. TaskCraft: Automated Generation of Agentic Tasks. In *International Conference on Learning Representations*, pp. 43714–43734, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/48644509339cb3076f7b0407c7588af6-Paper-Conference.pdf.

Wenhang Shi, Jinhao Dong, Yiren Chen, Zhe Zhao, Shuqing Bian, Wei Lu, and Xiaoyong Du. Scaling agentic capabilities via grounded interaction synthesis, 2026b. URL https://arxiv.org/abs/2606.02001.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Huatong Song, Fei Bai, Ming Yang, Renyuan Li, Jia Deng, Jujie He, Zhange Zhang, Daixuan Cheng, Yan Xing, Qi Yun, Xuxing Chen, Danyang Li, Feng Chang, Chuan Hao, Ran Tao, Jian Yang, Bryan Dai, Wayne Xin Zhao, Mingjie Tang, and Ji-Rong Wen. ClawGym II: Exploring black-box RL on agent harness, 2026a. URL https://arxiv.org/abs/2608.16798.

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. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8326–8357. Association for Computational Linguistics, 2026b. doi: 10.18653/v1/2026.findings-acl.407. URL https://aclanthology.org/2026.findings-acl.407/.

Michael Sullivan, Mareike Hartmann, and Alexander Koller. Procedural Environment Generation for Tool-Use Agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 18544–18562. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.936. URL https://aclanthology.org/2025.emnlp-main.936/.

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. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

Dunwei Tu, Hongyan Hao, Hansi Yang, Yihao Chen, Yi-Kai Zhang, Zhikang Xia, Yu Yang, Yueqing Sun, Xingchen Liu, Furao Shen, Qi Gu, Hui Su, and Xunliang Cai. ScaleEnv: Scaling environment synthesis from scratch for generalist interactive tool-use agent training, 2026. URL https://arxiv.org/abs/2602.06820.

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. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=P8E4Br72j3.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill

Qian, Daniel Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. In *International Conference on Learning Representations*, pp. 65882–65919, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/a4b6ad6b48850c0c331d1259fc66a69c-Paper-Conference.pdf.

Zezhong Wang, Xingshan Zeng, Weiwen Liu, Liangyou Li, Yasheng Wang, Lifeng Shang, Xin Jiang, Qun Liu, and Kam-Fai Wong. ToolFlow: Boosting LLM Tool-Calling Through Natural and Coherent Dialogue Synthesis. 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. 4246–4263. Association for Computational Linguistics, 2025b. doi: 10.18653/v1/2025.naacl-long.214. URL https://aclanthology.org/2025.naacl-long.214/.

Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. Agent World Model: Infinity Synthetic Environments for Agentic Reinforcement Learning. In *International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=OvLnVkV14P.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning, 2025c. URL https://arxiv.org/abs/2504.20073.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. In *Advances in Neural Information Processing Systems*, volume 37, pp. 52040–52094. Curran Associates, Inc., 2024. doi: 10.52202/079017-1650. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5d413e48f84dc61244b6be550f1cd8f5-Paper-Datasets_and_Benchmarks_Track.pdf.

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, 2026a. URL https://arxiv.org/abs/2605.18703.

Zhangchen Xu, Yuetai Li, Fengqing Jiang, Bhaskar Ramasubramanian, Luyao Niu, Basel Alomair, Bill Yuchen Lin, and Radha Poovendran. TinyV: Reducing False Negatives in Verification Improves RL for LLM Reasoning. *Transactions on Machine Learning Research*, 2026b. URL https://openreview.net/forum?id=HMGsqApBM3.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652. Curran Associates, Inc., 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf.

John Yang, Kilian Lieret, Carlos Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling Data for Software Engineering Agents. In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025. doi: 10.52202/085713-3239. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/8b86cf5ace600c48fd188efbb8dedec8-Paper-Datasets_and_Benchmarks_Track.pdf.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference*

on Learning Representations, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. In *International Conference on Learning Representations*, pp. 9965–10017, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b126cc38b8638e07bef37e7b2bb72bf-Abstract-Conference.html.

Yilun Yao, Xinyu Tan, Chao-Hsuan Liu, Yaoming Li, Zhengyang Wang, Wenhan Yu, Zhewen Tan, Yuxuan Tian, Guangxiang Zhao, Lin Sun, Xiangzheng Zhang, and Tong Yang. Harness-Bench: Measuring harness effects across models in realistic agent workflows, 2026. URL https://arxiv.org/abs/2605.27922.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, Qi Liu, Zhifang Sui, and Tong Yang. Claw-Eval: Towards trustworthy evaluation of autonomous agents, 2026. URL https://arxiv.org/abs/2604.06132.

Junkeun Yi, Damon Mosk-Aoyama, Baihe Huang, Ritu Gala, Charles Wang, Sugam Dipak Devare, Khushi Bhardwaj, Abhibha Gupta, Oleksii Kuchaiev, Jiantao Jiao, Jian Zhang, and Venkat Srinivasan. PivotRL: High accuracy agentic post-training at low compute cost, 2026. URL https://arxiv.org/abs/2603.21383.

Fan Yin, Zifeng Wang, I-Hung Hsu, Jun Yan, Ke Jiang, Yanfei Chen, Jindong Gu, Long Le, Kai-Wei Chang, Chen-Yu Lee, Hamid Palangi, and Tomas Pfister. Magnet: Multi-turn Tool-use Data Synthesis and Distillation via Graph Translation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 32600–32616. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1566. URL https://aclanthology.org/2025.acl-long.1566/.

Zhiyuan Zeng, Hamish Ivison, Yiping Wang, Lifan Yuan, Shuyue Stella Li, Zhuorui Ye, Siting Li, Jacqueline He, Runlong Zhou, Tong Chen, Chenyang Zhao, Yulia Tsvetkov, Simon Shaolei Du, Natasha Jaques, Hao Peng, Pang Wei Koh, and Hannaneh Hajishirzi. RLVE: Scaling Up Reinforcement Learning for Language Models with Adaptive Verifiable Environments. In *International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=DRqjMnDQGq.

Ziyun Zhang, Zezhou Wang, Xiaoyi Zhang, Zongyu Guo, Jiahao Li, Bin Li, and Yan Lu. InfiniteWeb: Scalable Web Environment Synthesis for GUI Agent Training. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28465–28492. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1313. URL https://aclanthology.org/2026.acl-long.1313/.

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*, 2024. URL https://openreview.net/forum?id=oKn9c6ytLx.

A Additional Method Details

A.1 Atomic-Task Construction and Certification

This section details the three checks used to construct atomic-task libraries in Section 3.1.2. Each task description records the intent, required inputs, ordered tool chain, and completion conditions.

E2E execution validation. Each test starts from an independently initialized environment state and executes MCP tools. A task passes when returned observations and persistent state satisfy its completion conditions. Passing tasks accumulate in V; test, response, and state diagnostics from failures accumulate in F for repair. Passing tasks are retained without regeneration.

Tool-chain orthogonality. Let ℓ_i denote the ordered tool-name sequence of task $a_i \in V$. Two tasks are redundant when either chain is a subsequence of the other:

$$\ell_i \preceq \ell_j \quad \text{or} \quad \ell_j \preceq \ell_i.$$

Here $\preceq$ preserves order but does not require contiguity, and includes equality. For example, $(u_1, u_3) \preceq (u_1, u_2, u_3)$. Filtering removes the task with the strictly contained chain; identical chains retain the earlier task. Sharing tools alone does not imply redundancy. Each round filters V to obtain the retained set A and redundancy feedback D.

Tool coverage. Coverage is the number of distinct MCP tools in the retained task chains divided by the MCP's total number of tools. We compute it after orthogonality filtering, using the declared tool chains of retained tasks. Let T denote the MCP tool set and T_a the tools in task a's chain. Uncovered tools guide further generation toward the target coverage η.

Incremental construction. Algorithm 1 combines these checks, using failure, redundancy, and coverage feedback in the ongoing synthesis conversation. The budget R includes the initial validation round. Construction stops when the target coverage is reached, the round budget is exhausted, or no new candidates remain. We return the retained tasks and attained coverage, and save the tasks, E2E tests, and validation reports.

Algorithm 1 Incremental Atomic-Task Construction and Certification

Input: Certified MCP m with tools T, state schema, and execution contracts; synthesis agent g, target coverage η, maximum validation rounds $R \geq 1$
Output: Filtered task set A and attained tool coverage c

$V \leftarrow \emptyset$; $F \leftarrow \emptyset$
$B \leftarrow \textsc{Synthesize}(g, m)$
for $r = 1, \ldots, R$ **do**
  $(P, F_r) \leftarrow \textsc{ValidateE2E}(m, B)$ ▷ Independent state per task
  $V \leftarrow V \cup P$; $F \leftarrow F \cup F_r$
  $(A, D) \leftarrow \textsc{FilterOrthogonal}(V)$
  $C \leftarrow \bigcup_{a \in A} T_a$
  $c \leftarrow |C|/|T|$
  if ($A \neq \emptyset$ and $c \geq \eta$) or $r = R$ **then**
    break
  end if
  $B \leftarrow \textsc{ReviseAndExpand}(g, m, V, F, D, T \setminus C)$
  Remove from B tasks whose identifiers already occur in V
  if $B = \emptyset$ **then**
    break
  end if
end for
return (A, c)

A.2 Cross-MCP Task Composition

The atomic-task library is unevenly distributed across MCPs and domains. To control the resulting task distribution, we use five difficulty profiles, *Simple*, *Standard*, *Moderate*, *Complex*, and *Expert*. Each specifies ranges for atomic-task count, MCP count, and domain breadth, together with a target corpus proportion. These dimensions control task length, the number of services to coordinate, and the range of business contexts involved.

Given the requested corpus size, largest-remainder allocation converts profile proportions into integer quotas. Within a profile's constraints, the sampler uses atomic-task capacity and historical usage to select domains and MCPs, favoring capabilities underrepresented in generated tasks. It ranks candidate combinations by shared entities, complementary operations, and functional redundancy. The profile mixture controls corpus-level difficulty, while coverage-aware selection broadens the capabilities represented within each profile.

Composition requires a surplus of candidate atomic tasks because not every local objective fits a shared scenario. For a selected MCP set M and target atomic-task count k, we require

$$\sum_{m \in M} c_m \geq \lceil \rho k \rceil, \qquad \rho > 1,$$

where c_m is the number of certified atomic tasks in MCP m and ρ provides a proportional capacity surplus. A semantic evaluation model checks whether the selected MCPs support a coherent shared objective. Accepted combinations count toward the profile quota and update usage statistics before an atomic-task set is sampled. Rejected combinations are resampled within the same profile.

A.3 Environment and Task Statistics

The corpus contains 8,172 executable MCPs, 64,755 tools, and 41,695 certified atomic tasks. Figure 7(a–c) shows domain coverage and per-MCP capacity. Software & DevOps is the largest domain, accounting for 22.6% of MCPs. Larger tool inventories tend to support larger atomic-task libraries.

The 11,884 composed tasks have a median length of 20 atomic-task turns (Figure 7(d–e)). Five construction-time difficulty profiles, Simple, Standard, Moderate, Complex, and Expert, specify increasing task length and breadth across MCPs and domains. Their distributions capture both extended tool interaction and cross-MCP coordination, with broader domain coverage generally involving more MCPs.

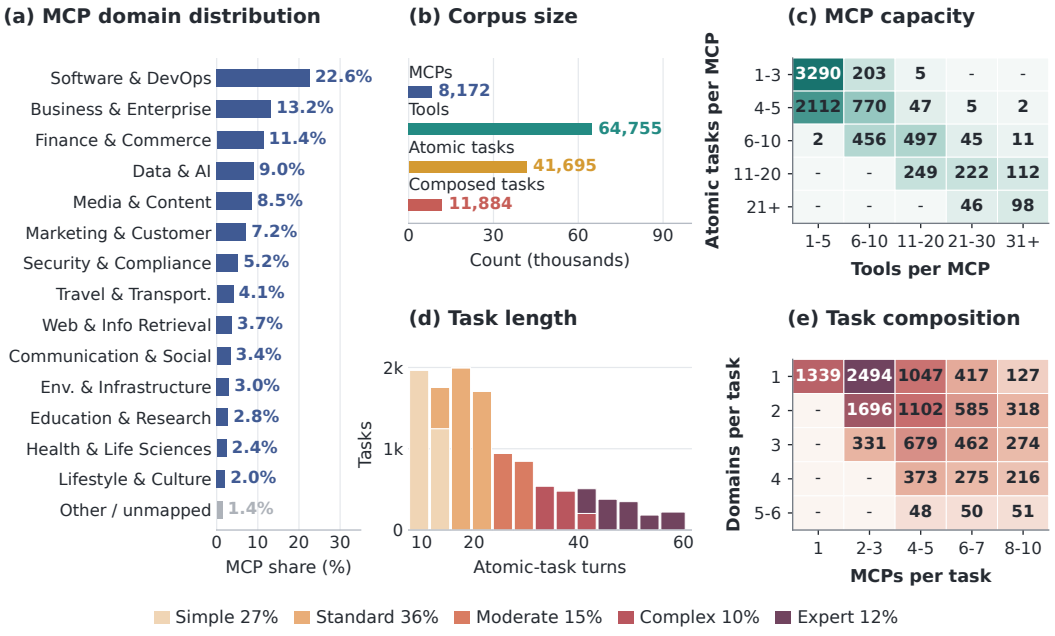


Figure 7: **Scale and composition of tool environments and agent tasks.** (a) MCP distribution across macro domains. (b) Total numbers of MCPs, tools, certified atomic tasks, and composed tasks. (c) MCP counts by tool and atomic-task inventories. (d) Atomic-task turns per composed task, colored by difficulty; legend percentages give the difficulty mix. (e) Task counts by the numbers of MCPs and primary domains. Heatmap cells report counts in the corresponding bins.

B Training and Evaluation Details

B.1 Training Configuration

SFT sampling. Kimi K3 (Kimi Team, 2026) generates SFT trajectories using prefix-preserving rejection sampling with up to $k = 3$ retries per atomic-task turn, allowing up to four attempts including the initial attempt.

RL task selection. We independently synthesize 5,000 candidate task–verifier pairs. For each task, Nex-N2-mini generates 16 pilot rollouts, which are independently scored by a rubric-based LLM judge and an executable verifier. We select pairs based on consistency between the two sets of scores, following Section 3.3.2. The resulting 1,954 tasks are used for both the 8B and 14B models.

RL optimization. The 8B and 14B models use atomic-turn GRPO with the optimization and rollout settings in Table 4. The reference policy is fixed at each model's RL initialization checkpoint.

Table 4: **RL training configuration.** Optimization and rollout settings for the 8B and 14B models.

Configuration	Setting
Optimizer	Adam
Learning rate and schedule	10^{-6}, constant
Adam moments	$\beta_1 = 0.9$, $\beta_2 = 0.98$
Weight decay / gradient clipping	0.1 / 1.0
Training precision	BF16
PPO lower / upper clipping	0.20 / 0.28 (ratio interval $[0.80, 1.28]$)
Reference KL	K2 estimator, coefficient 10^{-3}
Reward-side KL / entropy coefficient	0 / 0
Loss aggregation	Global token mean
Sampling temperature	1.0
Top-p / top-k	1.0 / disabled
Maximum context length	131,072 tokens; no additional fixed per-turn generation cap
Maximum assistant turns	100
Token interface	Token-in/token-out (TITO)
TIS mode	Token-level weighting with out-of-range masking
TIS acceptance interval	$[0.5, 2.0]$; no batch weight normalization
Dynamic sampling	Complete groups with reward standard deviation $> 10^{-6}$
Infrastructure failure retries	At most two physical attempts per logical rollout

Token alignment and importance sampling. TITO preserves sampled token IDs and token-level rollout log-probabilities for trainable model outputs. The training backend consumes these token sequences without retokenizing the complete trajectory. Prompts, tool observations, inserted template tokens, and model-specific boundary replacements are excluded from the training loss. Token-level truncated importance sampling (TIS) weights the policy loss by the ratio between training-backend old-policy and rollout-policy probabilities. We retain weights in $[0.5, 2.0]$ and mask tokens outside this interval rather than clipping their weights to its endpoints. The TIS mask also applies to the KL term, while the loss denominator remains the original valid-token count.

Selective credit masking. We detect format and tool-interface errors, repeated tool interactions without progress, and excessive within-turn repetition of text or tool calls. For a flagged assistant turn, we set positive advantages to zero on all model-generated tokens while retaining negative advantages. Original rewards, subsequent recovery turns, and the KL term remain unchanged.

Dynamic sampling and failure handling. Dynamic sampling discards incomplete or constant-reward groups and samples new groups to replenish the training batch. Physical retries address recoverable infrastructure failures, not low-reward outcomes.

B.2 Evaluation Details

The main comparison and RL study use BFCL V4, τ^2-Bench, and Claw-Eval. The data-construction ablations use Toolathlon-Verified, AutomationBench, and Claw-Eval. Evaluation follows each benchmark's native harness and official aggregate metric.

External baseline results. The BFCL V4 and τ^2-Bench scores of Qwen3.5-35B-A3B in Table 1 are taken from its official model card. Its Claw-Eval average is reported in the Qwen3.5-35B-A3B comparison column of the official Qwen3.6-35B-A3B model card. For τ^2-Bench, Qwen follows the official setup except in the airline domain, where it applies the fixes specified in the Claude Opus 4.5 system card.

C Additional Experimental Results

C.1 Credit-Assignment Granularity

With SFT initialization, task pool, reward verification, and positive-credit masking fixed, we compare trajectory-level and atomic-turn GRPO. The former assigns one advantage from the trajectory's mean checkpoint reward to all turns; the latter computes advantages within atomic-turn groups from binary task rewards. Atomic-turn GRPO improves every benchmark at both scales (Table 3).

C.2 Learning Dynamics

Reward reporting. For the 14B model, each training step samples 16 tasks. For reward reporting, each task contributes 16 complete root-to-leaf trajectories reconstructed from the final rollout layer, giving 256 trajectories per step.

Learning curves. For WEFT-14B, mean reward rises from 0.493 to 0.584 as policy entropy decreases from 0.816 to 0.432 (Figure 8). We reconstruct complete trajectories by backtracking from each leaf in the final rollout layer to the root. Each trajectory's reward is the mean of its binary checkpoint outcomes (Equation 1). We average these rewards within each task group and then across task groups in the batch. This reported statistic reflects completion over the full task; training advantages use the current atomic task's binary reward within each atomic-turn group.

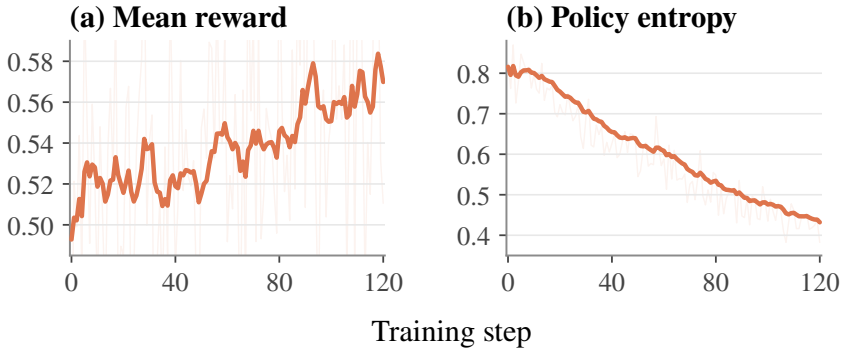


Figure 8: **Learning dynamics of WEFT-14B.** (a) Mean trajectory reward; (b) policy entropy.

D MegaMCP Runtime and Evaluation

D.1 Runtime and Deployment

MegaMCP implements the execution model in Section 3.3.3 through a versioned MCP registry, shared worker processes, and task-scoped session groups (Figure 9). The runtime manages state initialization, idle-session offloading, and recovery across workers.

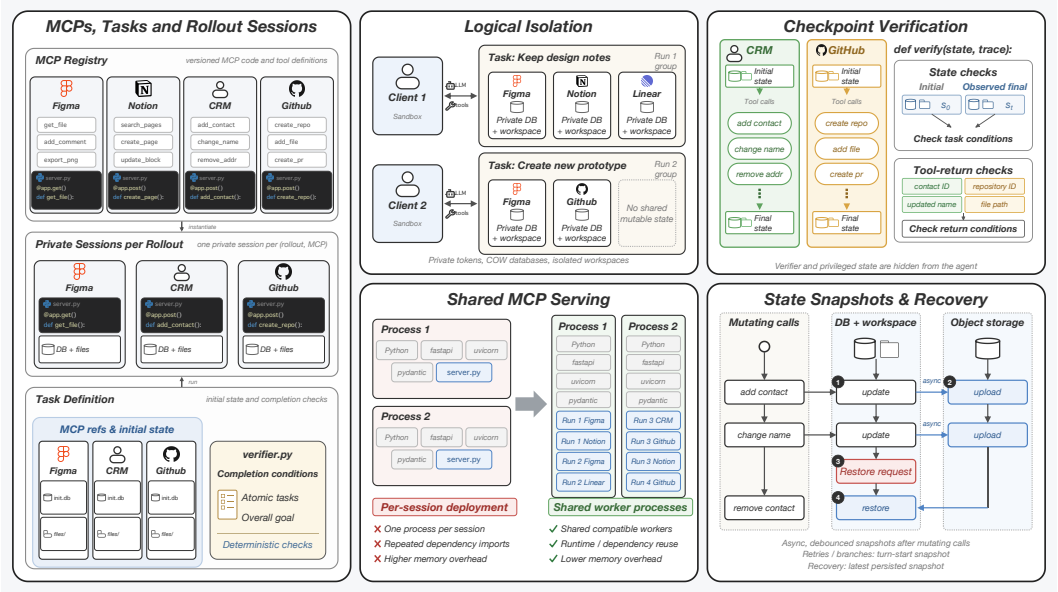


Figure 9: Overview of MegaMCP. A task's constituent MCPs are instantiated as private sessions within a unified lifecycle group. Compatible sessions share Python processes while retaining isolated mutable state. Initial and final databases and the complete tool-call ledger provide verification evidence, while persistent checkpoints support recovery and reloading.

Registry and process sharing. A training batch can require thousands of concurrent sessions across more than a thousand distinct MCPs. A registry stores MCP metadata, source code, and tool definitions as content-addressed records with monotonically increasing versions. Static analysis screens for process-global side effects and assigns a load class that limits how many instances share a process. Other servers use stronger isolation tiers. A prefork master preloads common web and database dependencies, then creates workers on demand. Each worker imports MCP servers as separate modules and binds requests to the requesting session's private database.

Private state and session lifecycle. A task references its MCPs by registry identifier and its initial databases by object-storage location, optionally including an executable verifier. Each run creates a session group containing one private session per MCP. Databases are copy-on-write clones of immutable initial snapshots, and mutable workspaces are isolated across runs. Pausing or resuming a task suspends or restores its MCP sessions and sandboxes as a group. Agents access server-side state through MCP tools; internal database files, server workspace paths, and credentials remain hidden. Each session has an individual access token.

Idle-session management. Agent generation leaves tool services idle between calls, with a median interval exceeding ten seconds. After a cooling window, MegaMCP offloads an idle session group's runtime while retaining its sessions, credentials, and workspaces. The next tool call reloads the group. Longer-idle groups are checkpointed and evicted from local disk, releasing resources between periods of active tool use.

Checkpointing and migration. After mutating calls, debounced asynchronous checkpoints persist consistent database and workspace snapshots to object storage under immutable, generation-fenced keys. Resumption and crash recovery load the latest persisted snapshot rather than replaying the full interaction. Workers monitor disk and memory pressure, stop accepting placements when unhealthy, and migrate affected sessions to healthy workers. Generation fencing prevents writes from workers that no longer own a session, and a reaper completes interrupted shutdowns idempotently. Persistent snapshots also support re-execution after environment or task revision.

Verification and operational feedback. MegaMCP retains each session's initial database, final database after a group-wide freeze, and full tool-call ledger with arguments, results, and outcomes. The verifier executes within the serving layer over these immutable artifacts without additional agent

instrumentation. Per-tool error rates and crash statistics are associated with the responsible MCP definitions, guiding server revalidation or stronger isolation in the execution-driven self-evolution loop (Section 3.2).

D.2 SERVING EVALUATION

We compare MegaMCP with local MCP serving inside E2B sandboxes using identical task traces. Local serving packages MCP source, initialized databases, and server processes into each sandbox. MegaMCP instead shares the servers while isolating mutable state by session. Both configurations use the same tool interfaces, initial states, planned calls, and end-of-task cleanup.

Workloads and timing boundaries. The cold-start comparison contains 100 tasks, 400 MCP sessions, and 3,791 planned tool calls, with sandbox creation and provisioning timed for both configurations. A larger workload contains 1,000 tasks, 4,000 sessions, and 39,864 planned calls, with up to 1,000 calls in flight. In this larger workload, local servers are prewarmed, while MegaMCP session provisioning is included in the timed interval. The cold-start comparison therefore measures startup under matched timing boundaries; the larger workload measures execution at rollout scale under the stated deployment conditions.

Table 5: **Data uploaded to agent sandboxes (MiB).** Workload data comprise the query, MCP source, and initialized databases; runner files contain the replay driver and dynamic configuration.

Workload	Serving path	Query	MCP source	Initdb	Workload data	Runner files	Total	Workload reduction
100 tasks, cold	Local	0.315	45.908	25.195	71.418	2.865	74.282	–
100 tasks, cold	MegaMCP	0.315	0	0	0.315	2.893	3.208	99.559%
1,000 tasks, warmed	Local	3.123	484.494	255.641	743.259	30.283	773.542	–
1,000 tasks, warmed	MegaMCP	3.123	0	0	3.123	31.717	34.841	99.580%

Provisioning data. MegaMCP removes repeated delivery of MCP source and initialized databases (Table 5). At 1,000 tasks, workload upload falls from 743.259 to 3.123 MiB, a 99.580% reduction; the 100-task comparison shows a similar 99.559% reduction. Including runner files, total upload decreases by 95.496% and 95.682%, respectively. Shared serving thus reduces data transfer while preserving the same task queries and private session state.

Table 6: **Serving latency.** TTFT is measured from the beginning of the timed interval to the first tool call; end-to-end latency additionally includes trace execution and cleanup. Latencies and wall time are in seconds.

Workload	Serving path	TTFT p50	TTFT p95	E2E p50	E2E p95	Wall time
100 tasks, cold	Local	10.455	14.460	24.271	79.673	332.469
100 tasks, cold	MegaMCP	4.794	6.198	18.609	71.006	321.383
1,000 tasks, warmed	Local	0.009	0.016	15.097	107.217	501.220
1,000 tasks, warmed	MegaMCP	39.281	69.983	74.056	175.964	587.394

Latency and lifecycle completion. Both configurations complete every task lifecycle. Under matched cold-start timing, MegaMCP reduces median time to the first tool call by 54.1%, median end-to-end latency by 23.3%, and total wall time by 3.3% (Table 6). At 1,000 tasks, MegaMCP completes all lifecycles but incurs higher latency than the prewarmed local servers, with its session provisioning still included in the measurement. The results distinguish the reduction in repeated data transfer from the remaining cost of provisioning concurrent sessions.

Resident memory. Figure 6(b) summarizes resident memory under process sharing. Each configuration is normalized to its own one-process-per-session baseline (Table 7). The 50-session configuration occupies 110 MiB in one process.

Table 7: **MegaMCP resident memory.** Relative memory is the fraction of the corresponding one-process-per-session baseline retained by MegaMCP.

Configuration	Processes	Relative memory (%)	Reduction (%)
52 server instances	8	22.4	77.6
50 sessions	1	4.3	95.7

Historical cold and warmed runs. Table 8 reports historical cold and warmed runs on the same set of 1,000 tasks. The cold-repeat bars in Figure 6(c) use the two rows named *Cold repeat*, whose recorded median TTFT values are 428.6 s for local serving and 17.4 s for MegaMCP. The complete primary comparisons and their timing boundaries are reported in Table 6.

Table 8: **Historical 1,000-task serving runs.** Wall time and median TTFT are in seconds; stages retain the names in the experiment records.

Stage	Serving	Wall (s)	TTFT p50 (s)
Cold baseline	Local	859.7	348.9
Early cold	MegaMCP	278.5	22.7
Stabilized cold	MegaMCP	548.1	22.2
Cold repeat	Local	968.9	428.6
Cold repeat	MegaMCP	550.3	17.4
First warmed	Local	500.7	1.278
First warmed	MegaMCP	625.9	39.7
Final warmed	Local	501.2	0.009
Final warmed	MegaMCP	587.4	39.3

E Extended Related Work

Environment Scaling. Executable environments support agent interaction with persistent state and evaluation against actual outcomes, as in WebArena, OSWorld, and AppWorld (Zhou et al., 2024; Xie et al., 2024; Trivedi et al., 2024). AgentGen, RandomWorld, and TaskCraft expand training resources through automated environment and compositional task generation (Hu et al., 2025a; Sullivan et al., 2025; Shi et al., 2026a). EnvScaler, ScaleEnv, and AWM further combine programmatic environment synthesis with task verification for post-training (Song et al., 2026b; Tu et al., 2026; Wang et al., 2026). Task diversity also extends beyond domain coverage to interaction structure: DIVE derives tasks from real tool executions (Chen et al., 2026a), while EnvFactory and GAIS construct complex interactions through topology-aware sampling and structured dependencies, respectively (Xu et al., 2026a; Shi et al., 2026b). Related efforts scale workspace tasks (Bai et al., 2026a), web environments (Bai et al., 2026b; Zhang et al., 2026), software engineering (Jain et al., 2025; Yang et al., 2025), and terminal tasks (Gandhi et al., 2026). For environment and task construction, WEFT composes execution-verified atomic tasks across MCPs. Dependencies, cross-system entity bindings, and initial state enable separate tool backends to support a shared long-horizon goal.

Tool-Use Agents. Tool-use research addresses both accurate API invocation and multi-step action guided by environmental feedback. Gorilla and ToolLLM expand the range of APIs available to models (Patil et al., 2024; Qin et al., 2024), while ReAct interleaves reasoning and action (Yao et al., 2023). User interaction is incorporated into evaluation and data generation by τ-bench, τ^2-Bench, ToolSandbox, and APIGen-MT (Yao et al., 2025; Barres et al., 2025; Lu et al., 2025; Prabhakar et al., 2025). The interaction interface also shapes agent performance: SWE-agent studies model-oriented computer interfaces (Yang et al., 2024). OpenHands and BrowserGym provide execution and evaluation frameworks (Wang et al., 2025a; de Chezelles et al., 2025), while Harness-Bench compares harness effects under shared tasks (Yao et al., 2026). Studies of interactive RL identify performance degradation under prompt formulations that differ from those used in training (Aissi et al., 2025). ClawGym II further explores mixed-harness training across heterogeneous execution systems (Song et al., 2026a). WEFT varies both task disclosure and execution harnesses, presenting the same task through autonomous execution or incremental user interaction while retaining each harness's native behavior. These variations share task goals, initial state, and verification conditions.

Self-Improvement. Self-improvement can change problem-solving behavior, the agent system, or the tasks used for training. Self-Refine iteratively revises outputs through self-feedback (Madaan et al., 2023), Reflexion uses verbal reflection to guide subsequent attempts (Shinn et al., 2023), and Voyager accumulates executable skills through continued exploration (Wang et al., 2024). At the system level, ADAS searches over agent programs (Hu et al., 2025b), while Agentic Harness Engineering uses execution traces to guide iterative harness changes (Lin et al., 2026). Training tasks can also adapt to policy capabilities: Agent-World generates tasks targeting capability gaps (Dong et al., 2026), RLVE adjusts problem difficulty (Zeng et al., 2026), and SPADE jointly trains an environment designer and a solver through self-play (Liu et al., 2026). EigenData coordinates the synthesis, auditing, and repair of databases, executable environments, and interaction trajectories (Chen et al., 2026b). WEFT uses execution traces and state evidence to distinguish policy failures on valid tasks from problems in environments, tasks, or verifiers. Attribution routes repairs to the corresponding components. Fresh rollouts test revisions and reveal new issues, sustaining iterative self-evolution of the agentic interaction system. Disagreements between executable verification and rubric-based assessment help identify verifier problems. Task-disclosure views and agent harnesses diversify the evidence used by this loop.

Agent Training. High-quality demonstrations underpin tool-use post-training. APIGen and ToolACE combine rule-based checks with execution or semantic verification (Liu et al., 2024; 2025). ToolFlow and Magnet use tool relationships and graph structure to synthesize coherent multi-turn dialogues (Wang et al., 2025b; Yin et al., 2025), while AgentRefine improves generalization through trajectories containing feedback and correction (Fu et al., 2025). WEFT performs rejection sampling at atomic-task boundaries, retaining successful prefixes with their database and workspace state and resampling only failed segments.

Reinforcement learning further optimizes policies from interaction outcomes. GRPO estimates advantages through within-group comparisons (Shao et al., 2024). RAGEN studies stability in multi-turn agent training (Wang et al., 2025c), while Agent Lightning decouples execution from training and decomposes trajectories into training transitions (Luo et al., 2025). Fine-grained credit assignment uses different comparison units. GiGPO groups actions by recurring states across trajectories (Feng et al., 2025). PivotRL performs local rollouts at informative intermediate turns in demonstrations (Yi et al., 2026), whereas BPO branches from sandbox snapshots and assigns credit using sibling terminal returns (He et al., 2026). Research on imperfect verifiers also identifies errors in executable reward signals (Cai et al., 2025b). WEFT selects training tasks using agreement between executable scores and semantic assessments, then samples candidates from identical histories and environment states. Group-relative credit is determined by the current atomic task's binary completion outcome. Selective positive-advantage masking prevents undesirable patterns within successful segments from being reinforced while preserving negative advantages and learning signals for subsequent recovery. MegaMCP supports concurrent demonstration sampling and RL through shared tool serving, isolated mutable state, and state recovery.