ECHOVERSE-MCP: STATEFUL TOOL ENVIRONMENTS FOR AGENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning for tool-using agents requires environments that maintain state, enforce application rules, and expose the consequences of actions. Building these environments by hand limits the range of workflows available for training. We present *Echoverse-MCP*, an LLM-assisted pipeline that combines public OpenAPI specifications with reusable domain models to construct isolated, stateful Model Context Protocol environments. Each environment pairs typed tools with a relational database, seeded records, and explicit state transitions. The pipeline generates multi-step tasks by executing reference programs and retaining their tool traces and database changes as evidence. The resulting resource contains 121 environments and 5,916 tools, with 19,041 tasks across 45 environments. Reference solutions average 7.15 calls and 6.24 distinct tools. A separate 600-task evaluation suite covers scripted-user and simulated-user interaction. Mixed GRPO post-training of Qwen3.5-4B raises reported BFCL accuracy by 5.6 percentage points and mean three-domain τ pass@1 by 3.2 points. The results support the use of generated executable environments for agent learning while showing that aggregate gains can coexist with regressions in individual capabilities.

1 INTRODUCTION

Language-model agents carry out work through sequences of tool calls: finding records, resolving missing information, and making changes that affect later actions. Refunding a payment, for example, requires locating the correct charge and checking whether its current state permits a refund. The quality of an agent's decision therefore depends on the application state as well as the syntax of its call. Reinforcement learning (RL) can train on these consequences, provided that the environment maintains state and supplies meaningful feedback throughout the interaction.

Constructing such environments is expensive. Existing executable benchmarks provide carefully designed tools, application state, and evaluation logic for a limited collection of domains (Lu et al., 2025; Trivedi et al., 2024; Yao et al., 2025). Model-based tool emulation can expand coverage at lower cost, but makes state transitions and failures dependent on another model (Ruan et al., 2024). RL places particular pressure on this trade-off: it requires repeated rollouts, diverse initial states, and feedback that distinguishes an invalid action from an environment failure. Generating more requests alone does not expand the states and workflows an agent can reach.

We introduce *Echoverse-MCP*, an LLM-assisted pipeline for constructing stateful tool environments from public OpenAPI specifications and reusable domain models. Public API specifications supply an interface, but an executable training environment also needs records, relationships, and rules governing how actions change them. The pipeline selects user-facing operations, expands the domain model, creates a relational schema and seeded state, and generates a Model Context Protocol (MCP) server. Tools execute against a private database and enforce status transitions, returning structured errors for invalid operations. Isolated state allows episodes to be reset without acting on live user accounts.

Echoverse-MCP currently contains 121 independently executable environments with 5,916 tools. We use these environments to construct tasks from executed reference programs, retaining the calls, responses, and database changes that support each request. The resulting collection contains 19,041 tasks across 45 environments. Tasks can be presented with *scripted users*, whose requests and follow-up messages are written in advance, or *simulated users*, who respond through a separate language

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.

model during the episode. These formats expose different demands: carrying out a fixed request and eliciting information through conversation.

We study the resource through a 600-task evaluation suite and mixed RL post-training. The selected Qwen3.5-4B run consumes 650 scripted-user and 630 simulated-user tasks. Its aggregate BFCL and three-domain τ scores exceed the reported baselines, although the improvements are uneven and the evaluation profiles require qualifications. Our goal is to provide executable training environments and traceable task evidence at broader application scale, while measuring which agent capabilities improve.

Our contributions are:

- **Scalable environment synthesis.** We introduce an end-to-end pipeline from public API specifications and reusable domain models to isolated MCP servers with persistent state and enforced application rules.
- **Execution-grounded task construction.** Reference programs connect natural-language requests to observed tool behaviour and database changes, with coverage feedback guiding subsequent generation.
- **Evaluation and mixed-interaction learning.** A 600-task suite measures scripted and simulated interaction separately, and mixed GRPO post-training evaluates the utility and limits of the generated tasks. We will release the evaluation suite with the camera-ready version.

2 Related Work

Tool-use data and execution-grounded supervision. API-Bank combines runnable APIs with dialogues that assess planning, retrieval, and tool invocation (Li et al., 2023). ToolLLM constructs ToolBench from real-world APIs, generated instructions, and searched solution paths (Qin et al., 2023). APIGen verifies generated examples through format checks, actual function execution, and semantic assessment (Liu et al., 2024b), while ToolACE synthesises API inventories and dialogues with rule-based and model-based verification (Liu et al., 2024a). These works establish the value of diverse tool-use data and checked solutions. Echoverse-MCP additionally constructs the persistent application state and transition rules against which reference programs execute, connecting task generation to an environment that can support repeated RL rollouts.

Executable and emulated tool environments. ToolEmu emulates execution with a language model, expanding tool-use and safety evaluation without implementing each backend (Ruan et al., 2024). Executable alternatives include ToolSandbox's stateful tools, implicit dependencies, user simulator, and milestone evaluation (Lu et al., 2025); AppWorld's interconnected applications and state-based evaluation (Trivedi et al., 2024); and τ-bench's policy-constrained agent–user interactions, assessed through final state and required responses (Yao et al., 2025). These systems establish the value of persistent state within curated application collections.

MCP-based evaluation. MCP-Universe evaluates agents through real-world MCP servers and live data sources (Luo et al., 2025). This complements our focus on generating isolated environments for training. MCP provides a common tool interface, but does not itself specify application state, reset semantics, or task references. Echoverse-MCP supplies these through database-backed implementations and recorded reference executions.

Environments for agent training. AgentGym supports heterogeneous environments through a common interface, using reward-filtered cross-environment fine-tuning and within-environment PPO (Xi et al., 2025). SWE-Gym turns repository issues into training data for policies and verifiers (Pan et al., 2025), while SWE-smith generates test-backed software-repair tasks over existing codebases (Yang et al., 2025). ToolRL studies reward type, scale, granularity, and scheduling for tool selection and application under GRPO (Qian et al., 2025). RAGEN studies multi-turn RL instability and motivates finer-grained feedback and diverse rollouts (Wang et al., 2025). Together, these studies show that executable tasks alone do not determine learning quality: reward design and rollout diversity also matter.

Our contribution is complementary: generating additional stateful tool environments, grounding their tasks in execution, and using scripted-user and simulated-user tasks together in a mixed training recipe.

3 METHOD

Echoverse-MCP first constructs executable, stateful MCP environments and then uses them to generate reference-grounded tasks in scripted-user and simulated-user formats. This section describes the environment artifacts, generation pipeline, executable validation, and task packaging process.

3.1 SYNTHETIC ENVIRONMENT CONSTRUCTION

Each environment contains an MCP server and a private SQLite database. Tools read and modify persistent records, so earlier actions affect the preconditions and results of later calls. The packaged artifacts comprise the server, an initial database snapshot, a tool specification, extracted status hints, and an approximate dependency graph.

Figure 1 summarises the construction pipeline. Each environment is independent, allowing environments for different services to be generated in parallel.

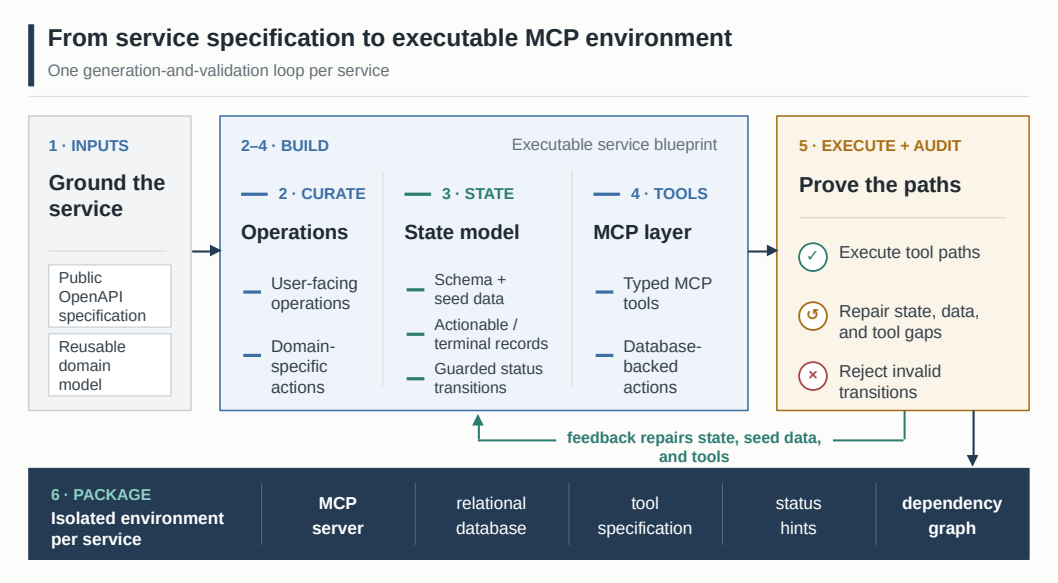

Figure 1: Echoverse-MCP environment construction. A public API specification and reusable domain model are curated into a relational state model and database-backed MCP tools. Tool-path execution feeds repairs back into the state, data, and tool implementation before an isolated environment is packaged with executable and machine-readable artifacts.

From API specification to executable state. We begin with a public OpenAPI specification and a reusable base environment. From the specification, we retain the operations most useful to an end user. Administrative, webhook, deprecated, and internal endpoints are removed, while domain actions such as *capture*, *finalize*, *approve*, and *refund* are prioritised. The curated operations and base domain model jointly determine the expanded database schema and initial data. Seeded records cover successful, actionable, and terminal states so that tasks can exercise more than the happy path.

From state to tools. We generate an MCP tool for each selected operation and connect it to the SQLite database. Tools include typed parameters, natural-language descriptions, and explicit error conditions. We then execute the available tool paths and fill data-coverage gaps. These checks identify empty read results and missing entities needed by state-changing tools, feeding repairs back into the implementation and seed data.

Business logic and validation. The environments enforce legal state transitions. For example, attempting to capture an already-refunded charge or finalize a posted invoice returns a structured error rather than a false success. The generation protocol prescribes checks that the server parses, artifacts can be extracted, representative tools execute, and an invalid state transition is rejected. It also extracts status preconditions and tool dependencies for downstream task generation.

Table 1 summarizes the resulting suite. All 121 environments provide the five packaged artifacts, and the tool count in the extracted specifications matches the count of decorated server functions. These artifact and execution checks establish concrete coverage; they do not constitute an exhaustive audit of application semantics.

Table 1: Statistics of the synthesised MCP environment suite ($N = 121$ environments).

Quantity	Total	Mean	Median	Range
Tools per environment	5,916	48.9	49.0	45–52
Dependency-graph edges	20,901	172.7	74.0	36–1,412
Guarded status preconditions	1,034	8.5	9.0	0–22

The dependency graph, which links a tool that produces an identifier to the tools that consume it, provides approximate scaffolding for generating multi-step tasks whose solutions require tool composition. Appendix B reports the distributions of tools, status hints, and dependency edges.

3.2 Execution-Grounded Task Generation

We generate tasks using a synthetic environment's tools and initial database state. The pipeline constructs a task outline, executes a Python reference solution, and uses the execution evidence to write the user-facing request (Figure 2). Accepted tasks guide subsequent generation toward underrepresented behaviours.

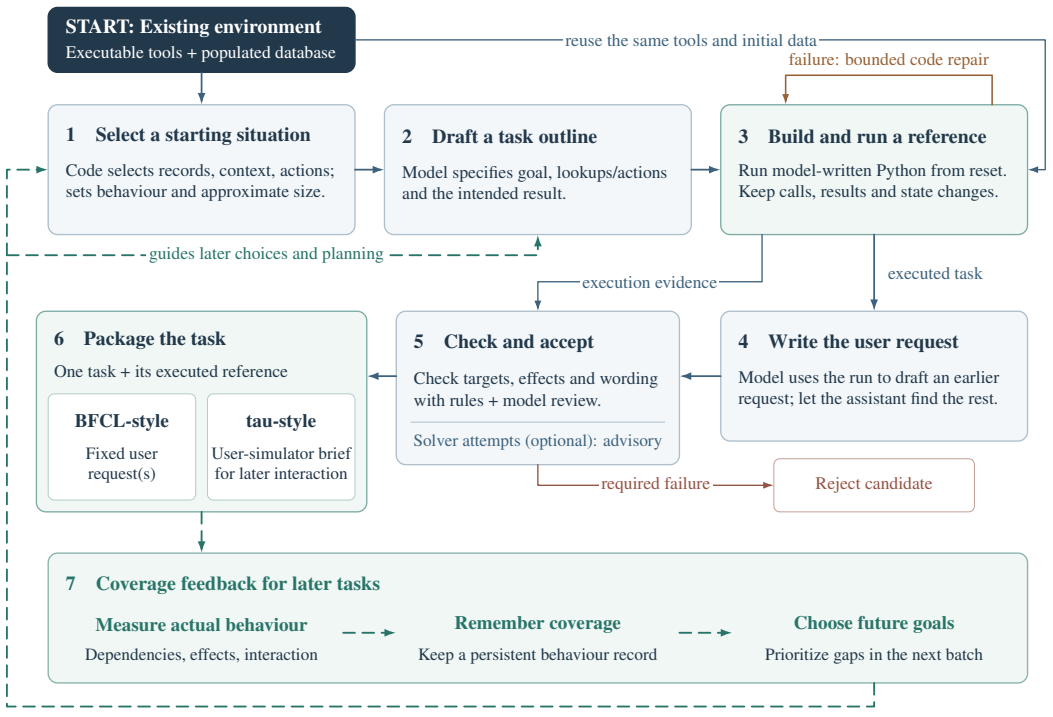

Figure 2: Execution-grounded task generation. Stages 1–6 select a starting state, draft and execute a reference, check the request, and package the task. Execution failures trigger bounded program repair. Stage 7 records accepted tasks' observed behaviour; the dashed feedback path guides later candidates toward gaps in coverage.

Constructing a task seed. The sampler selects an anchor record, such as a user, account, or project, and gathers related records through database relationships. It identifies possible actions

and chooses the required behaviour and approximate amount of work, potentially favoring tasks that are rare in the collection. If no suitable task is available, it can try another anchor.

Writing the task outline (TaskSpec). An LLM converts the seed into a *TaskSpec* describing relevant records, necessary lookups, intended actions, and expected outcomes. For example, a task may require finding an issue's milestone, creating a follow-up issue under it, and using the new issue's identifier to add a comment. The outline is checked against the database and tools, with bounded model corrections.

Writing and running a reference solution. An LLM turns the TaskSpec into a Python program that uses the environment's tools rather than modifying the database directly. We execute it against the initial database, recording arguments, responses, errors, and data changes. Crashes or deviations from the plan trigger bounded repair attempts. Intended tool errors are permitted: the aim is to carry out the specified behaviour, not eliminate every failed call. An execution that passes these checks becomes the reference solution.

Writing the request and checking the task. A language model uses the TaskSpec and execution summary to write a request describing what the user could have asked *before* execution. It supplies the goal and facts the user could reasonably know, leaving necessary discoveries to the assistant. Automatic checks examine whether the request exposes internal database identifiers and whether database changes satisfy the TaskSpec's outcome conditions. A separate reviewer checks alignment with the executed work, sufficient information, and whether evaluation can catch mistakes. Automatic findings and model judgments are recorded separately.

Preparing tasks for different interaction settings. Accepted tasks retain their requests, reference tool calls, database changes, and assessments. Scripted-user presentations provide a complete request or prewritten user messages, following a BFCL-style interaction format. In the τ-style simulated-user setting, a separate model receives instructions describing the user's background, goal, opening request, and information they do not yet know, then interacts during evaluation rather than generation. These are alternative presentations of the same task, not additional tasks.

Choosing what to generate next. We record each task's environment, response type, database changes, and interaction setting, and estimate connections between tool calls by matching earlier return values to later arguments. Before each batch, underrepresented categories guide the task type and approximate size and, when enabled, model preferences. Guidance does not guarantee the outcome: completed tasks are measured by their observed behaviour, which informs subsequent batches. Appendix C.1 provides additional generation details.

4 Task Corpus and Evaluation Suite

4.1 Generated Task Corpus

The saved task pool contains 19,041 tasks across 45 environments, including issue tracking, messaging, commerce, and financial services. Reference solutions average 7.15 tool calls and 6.24 distinct tools per task. The pool supports BFCL-style requests and τ-style simulated-user interactions; each task is counted once across formats. Appendix C.2 reports tool-use diversity, interaction-format coverage, and user-scenario details. These are corpus counts, distinct from the tasks consumed during training and the evaluation suite below. Reference calls describe executed solutions, not a lower bound on the number of calls required to solve a task.

4.2 Test Set Overview

The evaluation suite contains **300 scripted-user tasks** and **300 simulated-user tasks** across **20 executable tool environments**. Each environment models a consumer service through a Python server and a private SQLite database. Both task sets use the same environments, organized into four domain categories of five (Table 2).

In the scripted-user setting, messages are fixed in advance and replayed in order; the user cannot answer clarification questions. In the simulated-user setting, a separate language model receives a user brief and responds during the episode. The scripted set is scored against reference database state; the simulated set uses its own judging protocol. These are different tasks and evaluation procedures, so we report each set separately and do not average their scores.

Table 2: Composition by domain and interaction style. ENVS and TASKS are counts, other entries means. The environment columns appear once per domain: both styles use the same environments.

				Per task			
Domain	Style	Envs	Tools/env	Tasks	Calls	Reads	Writes
Gaming	scripted	5	50.8	75	9.83	6.28	3.55
	simulated			76	10.74	7.61	3.13
News & Publishing	scripted	5	49.4	76	9.89	6.33	3.57
	simulated			77	10.48	7.19	3.29
Sports, Fantasy & Betting	scripted	5	49.8	75	10.35	6.69	3.65
	simulated			74	11.14	7.66	3.47
Utilities	scripted	5	49.2	74	10.03	6.39	3.64
	simulated			73	10.62	7.11	3.51
All domains	scripted	**20**	**49.8**	**300**	**10.02**	**6.42**	**3.60**
	simulated			**300**	**10.74**	**7.39**	**3.35**

At every turn the agent receives all 47–52 tools in its environment. Scripted and simulated references use 8.90 and 9.19 distinct tools on average, respectively, making tool selection part of the task. Scripted references average 10.02 calls, of which 64.1% are reads: agents must locate relevant records before changing them. Every scripted task produces a database delta. Appendix F gives the workload distributions, capability coverage, grading details, and task-selection qualifications.

Workload and state changes. Most scripted tasks span three to five user turns, and all produce a reference database change. Figures 3 and 4 show the episode lengths, tool-use workload, and balance between reads and writes. These describe reference executions and the observed conversations, not minimum solution lengths or a controlled comparison of user formats.

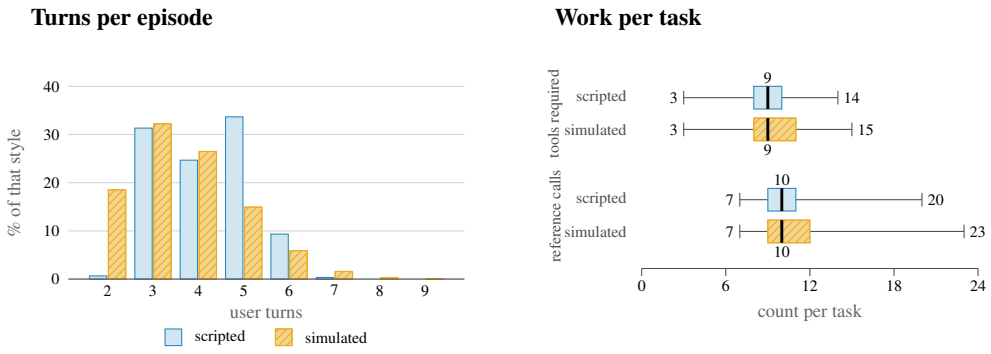

Figure 3: Episode length and reference workload. Left: user-turn distributions for 300 scripted tasks and 1,896 naturally completed simulated-user episodes, each normalized within its own population. Right: distinct tools and calls per reference. Boxes span the interquartile range, center marks show medians, and whiskers show the observed ranges.

5 EXPERIMENTS

Our experiments examine whether RL post-training using our synthetic environments improves tool use beyond the training tasks. We evaluate function-calling accuracy on BFCL (Gorilla Project, 2025), conversational task completion in the three-domain τ evaluation (Tau3), and performance on the Echoverse-MCP suite. These evaluations measure different capabilities and retain their own interaction and grading protocols.

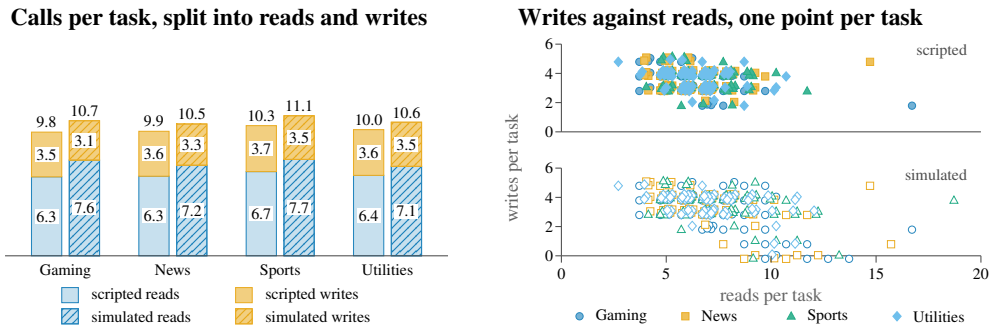

Figure 4: Read and write calls in reference executions. Left: mean calls by domain and interaction format, split into reads and writes. Right: per-task counts, with jitter to separate overlapping points. Scripted and simulated tasks are shown separately. These measurements characterize the selected reference programs, not a causal effect of the user-interaction format.

5.1 Training Setup

We initialize *Qwen3.5-4B* from its base checkpoint and post-train it with group relative policy optimization (GRPO; Shao et al., 2024) on reference-grounded synthetic tasks in executable environments. The mixed run combines Rich scripted-user and Tau simulated-user tasks; a fixed *Qwen3.8-27B* model acts as the simulated user and supplies model-based reward judgments, while only the policy is updated. Training consumed 1,280 distinct tasks: 650 Rich and 630 Tau, with 12 policy rollouts per task and 15,360 trajectories in total. Rewards combine executable state checks with model-based judgments. Appendix A provides the interaction variants, exposure accounting, and reward components.

5.2 Results and Analysis

Checkpoints are selected using separate validation measurements, not the external benchmark scores. Tables 3 and 4 report all BFCL and Tau3 conditions. The no-thinking Qwen3.5-4B Mixed RL checkpoint is matched across benchmarks; other artifact and prompt-profile qualifications are in Appendix D.

Table 3: Full BFCL results (percent). All rows use category-aware prompts. Prompt-profile and partial-rerun qualifications are given in the text and Appendix D.

Model	Training	Thinking	Overall	Non-live AST	Live	Multi-turn	Memory
Qwen3.5-4B	Base	Off	51.34	77.73	77.17	41.12	37.42
Qwen3-8B	Base	Off	40.55	85.71	77.92	21.62	23.01
Qwen3.5-4B	Mixed RL	Off	56.93	76.33	73.89	50.50	45.16
Qwen3-8B	Mixed RL	Off	38.85	86.50	78.49	21.00	18.92

External transfer is uneven across capabilities. Under the reported BFCL profile, mixed post-training raises overall accuracy from 51.34% to 56.93%. The 9.38-point multi-turn and 7.74-point memory gains are consistent with stronger sustained execution and reuse of context, while non-live AST and live accuracy decline by 1.40 and 3.28 points. Category-aware-2 directly evaluates only 155 memory-vector examples; the remaining 4,751 are inherited from each checkpoint's Category-aware run, so this is not a full direct prompt-matched evaluation.

For no-thinking Qwen3.5-4B, Tau3 pass@1 rises from 76.0% to 85.0% on airline, remains at 77.9% on retail, and rises from 98.2% to 98.9% on telecom, a mean gain of approximately 3.2 points. The base uses five-run cellwise minima, not typical-run performance. Other conditions use different artifacts across BFCL and Tau3 and are not combined into cross-benchmark gains (Appendix D).

Performance on the generated evaluation suite. Table 5 reports pass@1 separately for the scripted and simulated task sets. GPT-5.6 Sol has the highest retained score in both, at 35.17% and

Table 4: Full three-domain τ results (Tau3), in percent. The user simulator and judge are Qwen3.8-27B. Each no-thinking base cell is the minimum across five complete profile-matched runs; the row need not correspond to one run. All reported rows disable thinking. Cross-benchmark artifact qualifications are retained in Appendix D.

Model	Training	Thinking	Airline			Retail			Telecom		
			pass@1	pass@4	pass^4	pass@1	pass@4	pass^4	pass@1	pass@4	pass^4
Qwen3.5-4B	Base	Off	76.0	90.0	60.0	77.9	91.2	62.3	98.2	100.0	93.0
Qwen3.5-4B	Mixed RL	Off	85.0	92.0	60.0	77.9	95.6	62.3	98.9	100.0	97.4
Qwen3-8B	Base	Off	20.0	34.0	8.0	52.6	79.8	22.8	40.8	72.8	8.8
Qwen3-8B	Mixed RL	Off	21.5	38.0	8.0	49.3	77.2	24.6	39.9	67.5	12.3

pass@1 reports single-attempt success; pass@4 requires at least one successful attempt out of four, whereas pass^4 requires all four attempts to succeed.

64.3%, followed by different model orderings in each setting. Among Qwen baselines, Qwen3.8-27B leads scripted tasks at 27.8%, whereas Qwen3.5-27B leads simulated tasks at 53.1%. Within Qwen3.5, larger variants score higher in both settings; this association does not isolate parameter count from other model differences.

Table 5: Performance on the Echoverse-MCP evaluation suite (percent). Scripted and simulated tasks use different protocols and are reported separately. Both result columns report pass@1. Qwen baselines disable thinking.

Model	Condition	Scripted user pass@1	Simulated user pass@1
GPT-5.6 Sol	Frontier	35.17	64.3
GPT-5.6 Luna	Frontier	31.58	49.7
Qwen3.8-27B	Base	27.8	48.4
Qwen3.5-27B	Base	24.5	53.1
Qwen3.5-9B	Base	22.9	49.8
Qwen3.5-4B	Base	19.3	40.2
Qwen3.5-4B	Mixed RL	21.3	51.0

GPT-5.6 Luna scores 31.58% on scripted tasks and 49.7% on simulated tasks. The two sets have different tasks and judges, so their score gaps cannot be attributed solely to interaction style. Moreover, 31 scripted tasks were selected after Qwen3.8-27B failures, limiting interpretation as an unbiased model ranking (Appendix F).

Figure 5 complements these single-attempt scores with four-attempt baseline evaluations. Without thinking, Qwen3.8-27B reaches 46.0% scripted pass@4, but succeeds in all four attempts on only 13.3% of tasks. These baseline measurements are separate from the trained-model comparisons in Table 5.

Mixed RL post-training raises Qwen3.5-4B's reported pass@1 from 19.3% to 21.3% on scripted tasks and from 40.2% to 51.0% on simulated tasks: gains of 2.0 and 10.8 percentage points. The base and trained estimates use four and two attempts per task, respectively; the available aggregates do not establish statistical significance.

Scripted-user tasks exercise sustained execution across fixed messages, whereas simulated-user tasks also require gathering information through dialogue. The mixed policy improves in both settings, but separate ablations are needed to assign gains to either training component.

Overall, the systematic variation in scores across the Qwen model sizes, together with the differences observed among the frontier models, suggests that the test set provides meaningful separation between models with different capabilities rather than producing uniformly saturated performance.

Behaviour in trajectories. Across 60 scripted-user trajectories, the policy satisfies 155 of 180 required checks (86.1%), and 58 trajectories (96.7%) avoid extra state changes. Of 36 simulated-user

	avg pass@1		
	scripted		simulated
Agent	on	off	off
● Qwen3.8-27B	31.9	27.8	48.4
■ Qwen3.5-27B	24.0	24.5	53.1
▲ Qwen3.5-9B	21.7	22.9	49.8
◆ Qwen3.5-4B	20.0	19.3	40.2

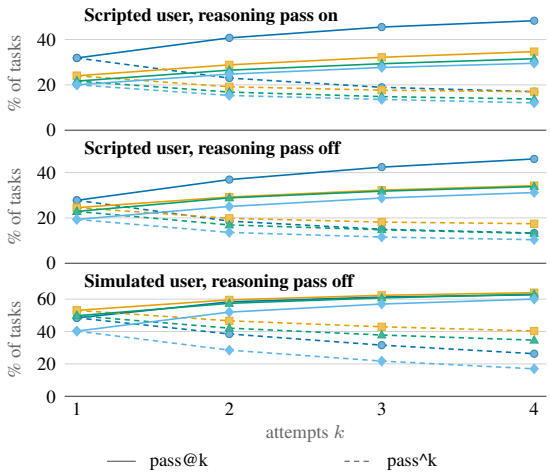

Figure 5: Baseline coverage and repeated reliability with four attempts per task. Solid curves show pass@k (at least one success); dashed curves show pass^k (all attempts succeed). The upper two panels use the same scripted tasks; thinking-on and thinking-off runs have different time limits. The bottom panel uses different simulated-user tasks and a different judge, on its own vertical scale. Full rates and protocol qualifications are in Appendix F.

trajectories, 35 (97.2%) reach the correct final state, 30 (83.3%) meet all communication requirements, and 29 (80.6%) meet both. Conversations average 2.94 user turns and 6.94 tool rounds. No invalid tool call or forbidden shortcut is observed in these 96 trajectories. This batch illustrates controlled multi-step work, but does not establish a pre/post-training behavioural change. Appendix E gives three concrete examples.

5.3 Scope and Limitations

Executable references make tool behaviour inspectable, but their validity still depends on generated application logic, request quality, and validation coverage. Dependency graphs and status hints remain approximate. Complete prompt-matched benchmark evaluations, representative baselines, and the remaining local post-training conditions are needed to strengthen the reported comparisons.

6 Conclusion

Echoverse-MCP connects public API specifications to executable, stateful tool environments and uses reference execution to ground task construction. The resource contains 121 environments, 5,916 tools, and 19,041 tasks across 45 environments, with scripted-user and simulated-user formats. Its central contribution is a scalable way to supply persistent state, application rules, and inspectable task evidence for agent learning.

Mixed GRPO post-training produces higher aggregate BFCL and three-domain τ scores under the reported comparisons, alongside regressions in individual capabilities and evaluation limitations. On the 600-task suite, the mixed Qwen3.5-4B checkpoint has higher reported pass@1 in both interaction settings than its base model. Together, these results motivate broader executable training resources and evaluation that preserves differences between tasks, interaction protocols, and the quality of their supervision.

ETHICS STATEMENT

Echoverse-MCP uses public API specifications and synthetic database records to construct isolated application environments. The environments do not connect to live services, execute actions on real user accounts, or require private user data. Service names and interface structure are used to study tool-use behaviour and do not imply affiliation with or endorsement by the corresponding service providers. Any public release must preserve the licences and attribution requirements of its source specifications and exclude credentials or other sensitive information.

Executable tool environments and agents trained on them are dual-use: the same capabilities that support benign workflow automation could be applied to unauthorised actions if connected to real systems. We therefore evaluate in sandboxed environments and recommend explicit authorisation, least-privilege credentials, human confirmation for consequential actions, and audit logging before deployment. Synthetic user profiles and model-generated tasks may also encode demographic or behavioural stereotypes; they should not be interpreted as representative samples of real users or used to make decisions about people.

AI USE STATEMENT

Generative AI systems are part of the research method described in this paper. They are used to propose candidate environment implementations, task specifications, reference programs, and user requests, and to provide simulated users and model-based reward judgments in the reported training setup. These outputs are not treated as ground truth: environment code and reference programs are executed, state changes are checked against the database, and mandatory programmatic checks take precedence over model judgments.

AI-assisted tools also supported software development, literature lookup, manuscript drafting and language editing, figure preparation, and consistency checks. The authors reviewed the resulting code, prose, citations, and reported measurements and remain responsible for the final manuscript and artifacts.

REPRODUCIBILITY STATEMENT

Each generated environment is packaged with an executable MCP server, an initial SQLite database, a machine-readable tool specification, extracted status hints, and an approximate dependency graph. Accepted tasks retain their environment identity and recorded reference execution, allowing the initial state, tool sequence, and final database changes to be inspected. The experimental tables explicitly mark averaged, partially inherited, or artifact-mismatched results rather than treating them as exact paired comparisons.

The accompanying artifact release will include the generation and validation code, pinned source specifications, environment and task data, training configurations, evaluation scripts, random seeds, and exact checkpoint identifiers needed to reproduce the reported runs. We will also document the versions of external models used as simulated users, judges, and reward models. Because frontier model behaviour may change, reproductions that depend on a third-party endpoint should record the provider, model identifier, request parameters, date, and raw per-example outputs in addition to aggregate scores.

REFERENCES

Gorilla Project. Berkeley Function Calling Leaderboard (BFCL). Benchmark and evaluation code, 2025. URL https://github.com/ShishirPatil/gorilla/tree/main/berkeley-function-call-leaderboard.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. *arXiv preprint arXiv:2304.08244*, 2023. URL https://arxiv.org/abs/2304.08244.

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. *arXiv preprint arXiv:2409.00920*, 2024a. URL https://arxiv.org/abs/2409.00920.

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. *arXiv preprint arXiv:2406.18518*, 2024b. URL https://arxiv.org/abs/2406.18518.

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, 2025. doi: 10.18653/v1/2025.findings-naacl.65. URL https://aclanthology.org/2025.findings-naacl.65/.

Ziyang Luo, Zhiqi Shen, Wenzhuo Yang, Zirui Zhao, Prathyusha Jwalapuram, Amrita Saha, Doyen Sahoo, Silvio Savarese, Caiming Xiong, and Junnan Li. MCP-Universe: Benchmarking large language models with real-world model context protocol servers. *arXiv preprint arXiv:2508.14704*, 2025. URL https://arxiv.org/abs/2508.14704.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-gym. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 47717–47737. PMLR, 2025. URL https://proceedings.mlr.press/v267/pan25g.html.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. ToolRL: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*, 2025. URL https://arxiv.org/abs/2504.13958.

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. *arXiv preprint arXiv:2307.16789*, 2023. URL https://arxiv.org/abs/2307.16789.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris Maddison, and Tatsunori Hashimoto. Identifying the risks of LM agents with an LM-emulated sandbox. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/7274ed909a312d4d869cc328ad1c5f04-Abstract-Conference.html.

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. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

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*, pp. 16022–16076, 2024. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

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. *arXiv preprint arXiv:2504.20073*, 2025. URL https://arxiv.org/abs/2504.20073.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Xin Guo, Dingwen Yang, Chenyang Liao, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. AgentGym: Evaluating and training large language model-based agents across diverse environments. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27914–27961. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1355. URL https://aclanthology.org/2025.acl-long.1355/.

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, 2025. doi: 10.52202/085713-3239. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/8b86cf5ace600c48fd188efbb8dedec8-Abstract-Datasets_and_Benchmarks_Track.html.

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*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b126cc38b8638e07bef37e7b2bb72bf-Abstract-Conference.html.

A Training Details

Protocol and interaction variants. We initialize Qwen3.5-4B from its base checkpoint and optimize only the policy with GRPO. Training draws from two task presentations derived from executable environments. Rich scripted-user tasks provide the complete request, including any prewritten follow-up messages, before or during the episode. Tau simulated-user tasks instead use a fixed Qwen3.8-27B model to play the user during the interaction. The same fixed model supplies model-based reward judgments. Thinking-mode generation is disabled in the reported runs. Reported artifacts are selected using separate validation measurements rather than external BFCL or τ-bench test scores.

Realized exposure. The reported count describes tasks actually consumed through update 160, not the cardinality of the source parquet or the full generated task pool. The 160 completed updates contain 1,280 distinct task IDs: 650 Rich scripted-user tasks and 630 Tau simulated-user tasks. Each task forms one prompt group with 12 policy rollouts, yielding 15,360 training trajectories. No task ID repeats within these updates.

Reproduction scope. Exact reproduction requires the optimizer configuration, reward coefficients, generation settings, and checkpoint-selection rule in addition to the model roles and exposure counts reported here. These settings are not specified in the present manuscript and must be provided with the training configuration.

Reward components. Rewards are assigned to completed trajectories and combine executable state checks with model-based judgments. Scripted-user tasks receive credit for matching the reference final state under strict and lenient criteria, together with a model-based task judgment; asking

the user a question incurs a penalty in this setting. Simulated-user tasks receive credit for exact database-state matching, communicating required information, and reaching a semantically correct final state. These components provide partial credit during training, whereas benchmark success is evaluated separately using each benchmark's reported metric.

B Additional Environment Statistics

The 121 environments span consumer services, enterprise software, finance, travel, media, healthcare, and developer platforms. Examples include Amazon, Airbnb, GitHub, Slack, Stripe, Spotify, Uber, and Epic FHIR. Table 6 provides distributional details omitted from the main text.

Table 6: Additional statistics for the 121 generated environments. Guard counts refer to status preconditions extracted into `hints.json`; dependency edges are heuristic rather than manually annotated.

Statistic	Environments
45–50 tools (target range)	108 (89.3%)
51–52 tools	13 (10.7%)
At least one extracted status guard	109 (90.1%)
At least five extracted status guards	91 (75.2%)
At least ten extracted status guards	53 (43.8%)
More than 100 dependency edges	47 (38.8%)
More than 500 dependency edges	10 (8.3%)

Tool counts are tightly concentrated: 6 environments contain 45 tools, 5 contain 46, 8 contain 47, 24 contain 48, 27 contain 49, 38 contain 50, 11 contain 51, and 2 contain 52. Extracted status guards have an interquartile range of 5–13 per environment, with a median of 9. Twelve environments have no extracted status guard; because guard extraction is regex-based, this indicates missing metadata rather than proving that their servers contain no state checks.

Dependency-graph size is more skewed. The first quartile, median, and third quartile are 62, 74, and 167 edges, respectively. The largest graphs occur in Instagram (1,412 edges), Twitch (1,305), YouTube (975), Figma (820), and Dropbox (696). These counts measure identifier connections recovered by the heuristic graph builder, not semantic task difficulty.

C Task Generation and Dataset Details

C.1 Task Generation Details

Seed context and task outline. The seed describes *who or what the task concerns, what information is available, and what actions might be possible.* In the GitHub environment, selecting a user may provide associated repositories and issues, with possible actions such as updating an issue or creating a new one. The sampler also chooses an approximate amount of work, such as successfully completing a request using one or two state-changing calls. The TaskSpec turns this starting situation into a concrete goal and records what the user could reasonably know.

Example reference task. A TaskSpec may propose creating a follow-up issue under the same milestone as an existing issue, then adding a comment explaining the connection. The reference program must first find the original issue and its milestone, create the follow-up issue, and use the identifier returned by the creation tool when posting the comment. Expected outcomes include whether the new issue belongs to the intended milestone. This distinguishes information available in the starting state from information obtained during execution.

Request formulation and review. Although the request writer sees the execution, the request must not reveal information that the assistant is expected to discover. In the example above, it can ask for a follow-up issue in the same milestone as an existing issue, leaving the assistant to look up that milestone rather than supplying its internal database identifier.

Successful execution alone does not establish that the wording is clear. A request to update "the latest issue," for example, must identify the relevant repository and make the ordering clear. The reviewer therefore assesses the request's information and evaluability as well as its alignment with the reference execution.

Coverage feedback. Coverage summaries distinguish, for example, tasks that create records from those that update records, and tasks that require looking up information before acting. Connections between calls are estimated by checking whether returned values, such as a newly created issue's identifier, appear in later arguments. When model guidance is enabled, selected categories supply preferences for the outline and subsequent generation stages. Accepted tasks are recorded according to their observed behaviour rather than their intended coverage target.

C.2 Dataset Details

Task content and variety. Table 7 summarizes the saved task pool. We measure tool-use variety by counting distinct combinations and sequences of tools separately within each environment. Changing only a record name or identifier does not create a new pattern. These measures capture structural variety, not necessarily differences in user goals or task difficulty.

Property	**Value**
Tasks	19,041
Environments	45
Distinct tools exercised	1,917
Largest environment's share of tasks	3.6%
Tool calls per reference (mean)	7.15
Distinct tools per reference (mean)	6.24
Distinct combinations of tools	14,981
Distinct ordered tool-call sequences	17,523

Table 7: Breadth and tool-use diversity of the saved task pool. Tools and call patterns are counted separately within each environment. Combinations ignore order and repetitions; sequences preserve both. Both ignore arguments and measure structural variety, not semantic uniqueness. Call statistics describe reference executions, not measured solver difficulty.

Interaction formats. Table 8 reports how many tasks support each format. BFCL multi-turn tasks average 3 user messages, authored during generation rather than produced by a live conversation. In the τ-style setting, the user model instead receives a brief explaining who the user is, what they want, and what they do and do not know. It responds dynamically during evaluation; live conversation lengths have not been measured.

Interaction format	Tasks	Pool share	Mean user turns	Range
BFCL single-turn	19,041	100.0%	1.00	1
BFCL multi-turn	11,772	61.8%	3.03	2–6
τ interactive	17,587	92.4%	Not measured	—

Table 8: Interaction formats. Counts overlap; shares use all 19,041 tasks. BFCL turns count prewritten user messages, not assistant replies or tool calls. Live τ conversation lengths have not been measured. The τ count excludes 1,454 tasks with neither a required database change nor a recorded answer value to check; it is not a finalized export count.

User scenarios and success criteria. The τ user scenarios include novice, intermediate, and expert users, with different communication styles, moods, and levels of verbosity. Success can require both completing an action and telling the user specific facts, such as a record's name or status. Other tasks require investigating a request and responding without changing the database.

D Evaluation Qualifications

Tables 3 and 4 in the main text report the updated benchmark results. Only the no-thinking Qwen3.5-4B Mixed RL row uses the same trained artifact across the two benchmarks. The Qwen3-8B variant uses different evaluated checkpoints across the two benchmarks and must not be combined into a paired cross-benchmark gain. The no-thinking Tau3 base uses cellwise minima over five runs for every metric, including pass@4 and pass^4. These are separate benchmark measurements, not the local Echoverse-MCP results reported in Table 5.

BFCL's no-thinking Qwen3.5-4B base and Mixed RL rows use Category-aware-2; the Qwen3-8B trained row uses Category-aware-3. The Qwen3-8B base profile is not specified separately. Category-aware-2 directly evaluates only 155 memory-vector examples and inherits the other 4,751 examples from the corresponding checkpoint's Category-aware run.

Table 5 records which local post-training results are reported. Each local comparison uses pass@1 from the retained table rows. Qwen baseline estimates use four attempts per task, the Mixed RL checkpoint uses two, and frontier models use one. Exact scored denominators and any exclusions for the frontier scripted rates are not specified in the supplied summary. No missing measurement is imputed from another run.

E Trajectory Examples

The following summaries retain the supplied requests, tool sequences, and reported outcomes. They illustrate lookup before mutation, explicit confirmation, and a decision to leave state unchanged. They are selected examples, not the basis for the aggregate rates reported in the main text.

Example 1: Scripted User (Cloudflare)

User	On `latticelabs.dev`, record the old apex IP, repoint it to `34.117.59.81`, add a proxied `app` record and cache rule, purge the cache, and never touch firewall rules. Report the old IP, final DNS-record count, and cache version.
Agent	Resolve the exact active zone before changing state.
Tool	`search_zones(latticelabs)`
Result	Selects active zone `latticelabs.dev` (ID 5), not the paused documentation zone.
Agent	Read the apex record and retain its previous value.
Tool	`list_zone_dns_records(zone=5)`
Result	Apex `@` points to `76.76.21.21`.
Agent	Execute only the requested DNS changes.
Tool	`update_dns_record` → `create_dns_record(app, proxied=true, ttl=120)`
Result	Both records point to `34.117.59.81`.
Agent	Verify the count, install the cache rule, and purge.
Tool	Second DNS listing → `create_page_rule` → `purge_cache(everything)`
Result	DNS count 2; page rule active; cache version 14.
Agent	**Final:** reports old IP `76.76.21.21`, record count 2, and cache version 14. No firewall tool is called.

Example 2: Simulated User (Dropbox)

User	**Turn 1:** Revoke partner link `https://dropbox.test/s/0018`, unlock "Henry Studio Wireframe Snapshot," and leave Abigail's team-2 folder share unchanged.
Agent	Ground both targets with read tools before requesting permission for the destructive actions.
Tool	`get_shared_link(0018)` → `search_files(Henry Studio Wireframe Snapshot)` → `get_file(18)`
Result	Link 18 is active; file 18 is the requested snapshot and is locked by user 18.
Agent	Asks the user to confirm exactly two writes and repeats that the team-2 share will remain untouched.
User	**Turn 2:** Confirms link revocation and file unlock; repeats that the team-2 share must remain active.
Agent	Execute only the confirmed mutations.
Tool	`revoke_shared_link(18)` → `unlock_file(18)`
Result	Link status `revoked`; `lock_owner_user_id=null`.
Agent	**Final:** reports both returned statuses and confirms that the unrelated folder share was not changed.

Example 3: Simulated User (Asana)

User	**Turn 1:** Move the due date and priority of "Kickoff brief for Q4 Revenue Dashboard," but only if the task belongs to the requester.
Agent	Read the target before deciding whether a write is authorized.
Tool	`search_tasks` → `get_task(1)`
Result	Assignee ID 4; creator ID 6; due date 2026-07-24; priority low.
Agent	Asks whether the requester is user 4.
User	**Turn 2:** Repeats that the task must remain unchanged unless it is hers and asks the agent to determine ownership.
Agent	Resolve the assignee, then inspect membership context rather than guessing the requester's identity.
Tool	`get_user(4)` → `list_team_memberships`
Result	User 4 is Victor Alvarez; membership data does not identify the requester.
Agent	Requests the missing identity explicitly.
User	**Turn 3:** Identifies herself as Zara Khan, states that she is neither assignee nor creator, forbids mutation, and asks for both owners' contact details and the current task state.
Agent	Keep the task unchanged and resolve the remaining owner.
Tool	`get_user(6)`
Result	Creator is Caleb Murphy (`caleb.murphy@asana.local`); assignee is Victor Alvarez (`victor.alvarez@asana.local`).
Agent	Reports no mutation, due date 2026-07-24, and priority low.
User	**Turn 4:** Requests one final confirmation that nothing changed.

Agent	Re-read the task rather than relying on the earlier snapshot.
Tool	`get_task(1)`
Result	Due date remains 2026-07-24 and priority remains low.
Agent	**Final:** confirms zero changes and repeats the assignee and creator details.

F Test Set

Section 4.2 describes the test set and the two interaction styles. This appendix reports workload, capability coverage, and evaluation details for the same 300 scripted and 300 simulated tasks.

F.1 Domain Coverage and Workload

The twenty environments fall into four categories of five (Table 2). Task counts are approximately balanced across categories: the largest contributes 77 of 300 tasks, or 25.7%. Environments within a category still have different schemas and tool signatures, so category balance does not imply equal task difficulty.

Each environment exposes 47–52 tools. This constrains variation in interface size, but tool semantics, initial state, and workflow requirements also affect difficulty. The agent receives the complete tool list at every turn.

Scripted references average 8.90 distinct tools and 10.02 calls per task, or 1.13 calls per distinct tool. Across all scripted references, 453 of the 996 available tools are exercised at least once. Simulated references use 9.19 distinct tools and 10.74 calls on average. These measurements describe the reference executions, not every valid solution.

F.2 Episode Length and Workload

Scripted requests reveal information through a fixed sequence of messages; 89.7% contain three to five user turns (Figure 3). The median reference uses 10 calls, and the reported mean is 2.53 calls per user turn. Distinct-tool counts are concentrated between 8 and 10. All 300 scripted references change the database, touching 2.87 tables on average. The grading target is the reference state, rather than an exact reproduction of the reference call sequence.

Simulated-user conversation lengths depend on the interaction and termination protocol. Across four evaluated models from 4B to 27B, episode means range from 3.35 to 3.75 user turns. The displayed distribution includes 1,896 episodes that ended naturally; it does not describe truncated episodes.

Scripted references average 6.42 reads and 3.60 writes, making reads 64.1% of all calls. Domain-level write means differ by 0.10 calls using the rounded table values, while read means differ by 0.41. Individual scripted references contain 2–5 writes and 3–17 reads (Figure 4). This wider read range reflects variation in the amount of record lookup in the saved solutions; it is not a controlled measure of search difficulty.

F.3 Behavioural Composition

Task selection covers eight recorded capabilities (Table 9). Labels overlap: 227 of 300 scripted tasks have at least one label, 86 have three or more, and labeled tasks average 2.05 labels. Multi-turn dialogue appears in 59.3% of tasks and deep write chains in 39.3%; every task with the latter label also has the former. The remaining 73 tasks have no recorded label, which does not imply that they lack these behaviours.

The simulated set is not labeled with this taxonomy. Seventeen tasks have special requirements: 12 call for declining a request, three require recovery from a tool error, and two require leaving the database unchanged. These small counts establish coverage, not reliable behaviour-specific rates.

Table 9: Agentic capabilities exercised by the scripted set. Tasks may carry more than one label; 227 of 300 carry at least one.

Capability	n	%	Capability	n	%
Multi-turn dialogue	178	59.3	Aggregate-then-act	31	10.3
Deep write chain	118	39.3	Write-after-write	19	6.3
Fan-in reconcile	76	25.3	Guarded create	5	1.7
Targeted update	38	12.7	Guarded delete	1	0.3

F.4 Evaluation Protocol

Table 5 contains the local evaluation results used in the analysis. The four Qwen baselines have four attempts per task; the Mixed RL checkpoint has two, and the frontier models have one. Every score in that table is pass@1. Scripted-user success requires an exact match with the reference database state; simulated-user success uses its separate judge. The two task sets are not pooled into a single score, and the available aggregates do not provide confidence intervals for the reported model differences.

Baseline repeated-attempt evaluation. Figure 5 and Table 10 report the separate four-attempt evaluations of the four Qwen base models. Pass@k measures at least one success among k attempts, whereas pass^k requires all k attempts to succeed. These results do not include the trained checkpoints in Table 5.

Table 10: Four-attempt baseline results (percent). Pass@k measures at least one success and pass^k requires all k attempts to succeed. Scripted success uses exact reference-state matching; its two conditions share tasks but differ in thinking and time limits. The simulated block uses different tasks and a separate judge. Results are compared within each block.

	Scripted, reasoning on					Scripted, reasoning off					Simulated, reasoning off				
	pass@k			pass^k		pass@k			pass^k		pass@k			pass^k	
Agent	1	2	4	2	4	1	2	4	2	4	1	2	4	2	4
Qwen3.8-27B	31.9	40.7	48.3	23.0	17.0	27.8	36.9	46.0	18.6	13.3	48.4	58.3	62.7	38.5	26.3
Qwen3.5-27B	24.0	28.8	34.7	19.2	17.0	24.5	29.2	34.2	19.8	17.4	53.1	59.6	64.0	46.6	40.3
Qwen3.5-9B	21.7	26.5	31.5	16.8	13.8	22.9	28.8	33.8	16.9	13.2	49.8	57.4	63.0	42.1	34.7
Qwen3.5-4B	20.0	24.7	29.5	15.3	12.1	19.3	25.1	31.2	13.6	10.4	40.2	52.0	60.0	28.5	17.0

Thinking and time budgets. The scripted thinking-on and thinking-off baseline runs use the same tasks, seeds, temperature, step budget, and judge, but per-task time limits are 900 and 180 seconds, respectively. Their comparison therefore does not isolate the effect of thinking. The simulated-user evaluations use a different task set and judge and are not pooled with scripted-user scores.

Exclusions and task selection. In each scripted thinking condition, ten of 4,800 Qwen baseline rollouts failed in transport and were excluded rather than scored as policy failures; none of the 4,800 simulated baseline rollouts were lost. Of the 300 scripted tasks, 31 were selected because Qwen3.8-27B had failed them. This model-informed selection limits interpretation as an unbiased ranking of agent capacity.

The reported scripted evaluations use a generic operations prompt that does not explicitly prohibit clarification questions, although the fixed user cannot answer them. Later no-clarification prompt results must be reported separately rather than combined with these measurements.