USERSCALER: BEYOND ENVIRONMENT SCALING — UNLOCKING THE USER DIMENSION FOR AGENT TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Scaling the training of large language model (LLM) agents relies on diverse and reliable environments. Existing methods leverage automatic pipelines to synthesize interactive, stateful, and scalable environments. Nevertheless, they overlook the substantial gap between idealized users and real-world users: realistic users often hold partial intents, exhibit variable phrasing styles, and react dynamically as interactions proceed. This discrepancy hinders the effectiveness of existing agents in real-world application. To tackle this issue, we propose **UserScaler**, which jointly generates plausible users and environments to bridge the gap between agent training and real-world deployment. UserScaler consists of three core components: (1) *Realistic user-centric environment synthesis*: a search agent constructs each environment following specifications of real services and derives user personas from market research of the target service; (2) *Realistic user-driven task generation*: user requests are synthesized conditioned on user personas and sampled toolchains; (3) *User-guided trajectory collection*: during rollout, simulated users actively engage in the interaction, whose behaviors are guided by an automatically derived rubric covering both process and outcome. We instantiate 209 environments (109 single-service and 100 cross-domain compositions), each equipped with roughly 50 user personas, yielding 12,073 training trajectories. Training Qwen3 models from 1.7B to 8B on this data consistently improves pass rates over corresponding baselines trained with EnvFactory, AWM, and EnvScaler across three agentic benchmarks: the 8B model reaches 44.3% on τ^2-bench, 19.8% on VitaBench, and 11.3 on UserBench. Under a matched budget of 6,474 trajectories, user-guided rollouts outperform direct ones, confirming that structured user modeling provides signal beyond additional data volume. We release the environments, training data, and models.

1 INTRODUCTION

Recent years have witnessed rapid progress in both the capabilities of LLM agents (Yao et al., 2023; Qin et al., 2024; Yang et al., 2024) and in environment synthesis for training them. Pipelines such as AgentScaler (Fang et al., 2025), AWM (Wang et al., 2026), EnvScaler (Song et al., 2026), EnvFactory (Xu et al., 2026), ScaleEnv (Tu et al., 2026), and AutoForge (Cai et al., 2025) now automatically construct hundreds to thousands of executable environments with rich tool interfaces. Yet a real-world LLM agent sits between two parties — an environment that provides tools and state, and a user who provides task intent — and existing pipelines systematically scale only the former: some entirely omit the user role (Wang et al., 2026; Song et al., 2026; Cai et al., 2025); some append a generic user prompt (Fang et al., 2025); and others generate ad-hoc personas disconnected from the training scenarios (Xu et al., 2026) (Table 1). This asymmetry matters: VitaBench (He et al., 2025) shows that agent success rates fluctuate by 30+ points across cooperative, anxious, and fragmented-information user profiles, and even the strongest models reach only $\sim$30% on cross-scenario tasks — a challenge particularly severe for smaller 1.7B–8B models that must learn to infer user intent from incomplete conversational contexts.

Given this gap, we aim to bring user modeling into training-data synthesis. However, directly injecting simulated users into existing pipelines faces two issues: (1) *Ungrounded personas*: users

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.

Table 1: Modeling of the user side across environment-synthesis pipelines and user-facing benchmarks. Environment counts show that quantity is no longer the bottleneck; the remaining columns show that the user side did not scale with it. ✓: supported, ○: partial, ✗: not supported.

	#Env	User in loop	Struct. persona	Persona grounding	Guided by rubric
User-facing benchmarks (evaluation)					
τ^2-bench (Barres et al., 2025)	3	✓	✗	hand-written	✗
VitaBench (He et al., 2025)	3 dom.	✓	✓	platform data	✗
User simulation (training data)					
PersonaForge (Lv et al., 2026)	—	✓	✓	seed queries	✗
Environment synthesis (training)					
AWM (Wang et al., 2026)	1,000	✗	✗	—	✗
EnvScaler (Song et al., 2026)	191	○	✗	—	✗
EnvFactory (Xu et al., 2026)	85	✓	✗	ad hoc	✗
ScaleEnv (Tu et al., 2026)	16 dom.	✓	○	per-task	✗
AutoForge (Cai et al., 2025)	10	✓	✗	—	✗
UserScaler (ours)	209	✓	✓	market research	✓

invented per-trajectory or drawn from small hand-crafted sets produce user–service mismatches that limit corpus coverage; (2) *Task-guidance dilemma*: the standard rubric-based filter systematically discards trajectories involving simulated users on complex tasks, leaving simple tasks overrepresented.

We propose **UserScaler**, a pipeline that scales training-time supervision along the user dimension via two components. **Grounded Persona Synthesis (GPS)** co-derives each environment and its $\sim$50 personas from the same evidence base (service specification + market research), structuring personas along *knowledge level* and *personality* axes to ensure realistic user–service pairings. **Rubric-Privileged User Guidance (RUG)** gives the simulated user privileged access to the task rubric and intervenes only when the agent stalls, converting failures on complex tasks into loss-masked supervision and shifting the corpus toward scenarios that stress the user dimension.

We instantiate 209 environments with $\sim$50 personas each, yielding 12,073 trajectories. Qwen3 models (1.7B–8B) trained on this data improve over the strongest prior method by +5.9 on τ^2-bench, +7.4 on VitaBench, and +0.4 on UserBench at the 8B scale; at a matched budget of 6,474 trajectories, user-guided rollouts outperform direct ones. Our contributions:

- **GPS**: the first environment-scaling method that jointly synthesizes environments and structured user personas from a shared evidence base, producing 209 environments with $\sim$50 personas each.
- **RUG**: a training-data collection mechanism where a rubric-informed simulated user converts complex trajectories that would otherwise be discarded into usable supervision.
- We release all 209 environments, 12,073 trajectories, and trained models, and demonstrate consistent gains across Qwen3 1.7B–8B on three agentic benchmarks with matched-budget controls isolating the user dimension as the source of gain.

2 Related Work

Environment scaling for tool agents. A growing line of work treats the environment, rather than the model, as the scarce resource in agent training (Luo et al., 2025). AgentScaler (Fang et al., 2025) abstracts function calling as read and write operations over databases, an abstraction that subsequent pipelines inherit: AWM (Wang et al., 2026) generates 1,000 SQL-backed environments with 35K tools through a five-stage scenario-to-verifier pipeline, EnvScaler (Song et al., 2026) synthesizes 191 environments programmatically from reusable skeletons, EnvFactory (Xu et al., 2026) samples topology-aware tool chains across 85 environments, ScaleEnv (Tu et al., 2026) builds a domain from

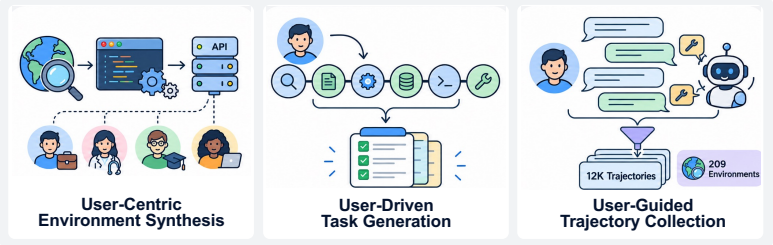


Figure 1: USERSCALER is a user-centric synthetic data pipeline that jointly generates environments and their plausible users, producing 12K training trajectories across 209 diverse environments for training tool-use agents.

scratch given only its name, and AutoForge (Cai et al., 2025) optimizes the generated environments themselves. Complementary data-centric efforts synthesize multi-turn tool-calling trajectories from existing APIs (Liu et al., 2024b;a; Wang et al., 2025a; Yin et al., 2025; Prabhakar et al., 2025), or mine real-world MCP servers (Xu et al., 2025). Quantity is no longer the bottleneck. What these pipelines scale, however, is the side of the interaction that the agent *calls*: tools, schemas, state transitions, and verifiers. The side that *issues* the request is left thin, and this shows up in the work itself — ScaleEnv observes that programmatic synthesis "struggles to construct complex, user-interactive tasks", while AutoForge introduces environment-level optimization specifically to suppress *noisy simulated users*, treating the user as interference rather than as signal. Table 1 makes this asymmetry explicit.

User simulation. Simulated users are well established in *evaluation*, where the user side has been pushed along two different axes. One is *capability*: τ^2-bench (Barres et al., 2025) models the episode as a dual-control process in which the user holds tools of its own and acts on the shared state, rather than merely answering questions — though each user is still specified by a hand-written instruction rather than by a persona. A third axis is *user-centricity*: UserBench (Qian et al., 2025) models underspecified goals where the agent must actively uncover latent preferences through multi-turn dialogue, reporting that even strong models fully align with all user intents only 20% of the time. The other is *profile*: VitaBench (He et al., 2025) derives personas from anonymized platform logs and reports that agent success drops sharply for anxious or scattered users relative to cooperative ones. Because leaking the user's latent preferences would invalidate a benchmark, this setting deliberately keeps the simulator uninformed — VitaBench 2.0 (Chen et al., 2026) restricts its user to a preset to-do list with minimal feedback. On the *training* side, users are far thinner: EnvScaler drives its user from a single generic system prompt and drops it entirely for RL (Song et al., 2026), EnvFactory generates an unstructured profile per trajectory (Xu et al., 2026), and ScaleEnv attaches a per-task constraint set of permissions, location, and history (Tu et al., 2026). PersonaForge (Lv et al., 2026) is closest to us, with a four-dimensional persona space of occupation, personality, proficiency, and background, but its personas are inferred backwards from real seed queries and are therefore decoupled from any environment. Raw persona count has never been the obstacle — Persona Hub (Ge et al., 2024) curates a billion of them — and personas are still typically handled as fixed descriptions (Chopra et al., 2026). What is missing is a persona population that is derived from the same evidence as the environment it inhabits, and that therefore grows as the environment set grows.

Supervision for long-horizon tasks. Synthesized tasks are useful only if trajectories can be obtained for them, and the prevailing recipe keeps successful rollouts and discards the rest. Recent RL-based approaches train agents to improve tool use through strategic reasoning (Feng et al., 2025; Guo et al., 2025) or multi-turn self-evolution (Wang et al., 2025b; Zhao et al., 2025), yet still require a base success rate to bootstrap from. TopoCurate (Yang et al., 2026) characterizes this family as "confined by an outcome-centric filtering paradigm where SFT trajectories are selected on binary success signals and RL tasks on pass-rate thresholds," and responds by projecting repeated rollouts into a semantic quotient topology to retain trajectories showing reflective recovery. A parallel line exploits privileged information offline: P2T (Ma et al., 2026) reconstructs trajectories from ground-truth patches behind a groundedness check, and PACT (Du et al., 2026) uses expert traces purely

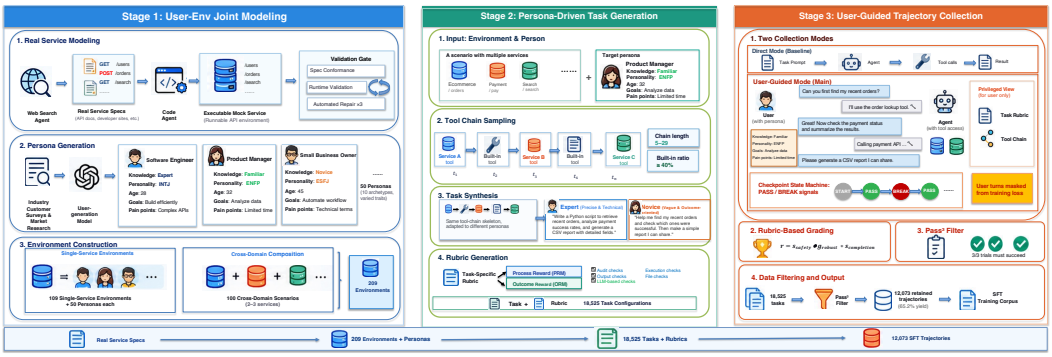


Figure 2: Overview of USERSCALER. Starting from user-centric environment synthesis, we progressively generate persona-driven tasks with rubrics, then collect user-guided multi-turn trajectories filtered by rubric-based scoring to obtain the SFT training corpus.

as an optimization-time signal, keeping rollouts prompt-only on the grounds that rollout-time hints induce a train–test mismatch. All of these operate *within* the distribution of trajectories the generator already produces; when a long-horizon task defeats every attempt, there is nothing to select from and nothing to reweight. We instead change that distribution, and the mismatch objection does not transfer: guidance reaches the agent through the user, who is present at deployment too, so an agent learning to exploit a clarification is learning an in-distribution skill rather than learning to be fed an answer — and the user turns are masked from the loss regardless.

3 USERSCALER

UserScaler synthesizes agent training data in three stages (Figure 2): it first constructs *environments* paired with the *users* who would plausibly use them (§3.1), then derives *tasks* whose feasibility is guaranteed by construction (§3.2), and finally collects *trajectories* under the guidance of the simulated user (§3.3). The recurring principle is that every artifact is grounded in something verifiable: environments in real service specifications, users in market research, tasks in executable tool chains, and rubric scores in the tool chain that produced the task.

Definitions. An *environment* is an executable service $e = (\mathcal{F}_e, d_e)$ where $\mathcal{F}_e$ is its set of API endpoints (tools) and d_e is its documentation derived from a real-world specification. A *persona* $p = (k, \pi, \mathbf{a})$ pairs a knowledge level $k \in \{\text{NOVICE}, \text{FAMILIAR}, \text{EXPERT}\}$ with a personality archetype π drawn from a pool of 10 archetypes and a set of domain-specific attributes $\mathbf{a}$ (occupation, goals, pain points, etc.). A *scenario* $s = (\{e_i\}, p_s)$ is a set of environments together with a primary persona. The pipeline maps a set of topics to a training corpus $\mathcal{D} = \{(\tau_j, r_j)\}$ of trajectories τ_j with scalar scores r_j. All generation steps use Qwen3.5-397B as the backbone LLM. Algorithm 1 in Appendix A summarizes the end-to-end flow.

3.1 USER-CENTRIC ENVIRONMENT GENERATION

3.1.1 GROUNDED ENVIRONMENT SYNTHESIS

Given a topic (e.g., "project management"), a **search agent** retrieves real service specifications — endpoint semantics, field schemas, and domain conventions. A **code agent** then materializes an executable mock environment from this evidence: it designs an API with 4–7 endpoints, generates a runnable server, and populates it with fixture data. Each environment undergoes a multi-level quality gate: specification conformance, runtime validation, and automated repair (up to three iterations). This loop yields **109** single-service environments.

To cover realistic multi-service workflows, a model further proposes **100 cross-domain scenarios** by selecting groups of 2–3 validated services linked by business logic (e.g., a CRM paired with an email service and a calendar), with the constraint that every constituent plays a functional role. One

service is designated as the **primary environment**, which anchors the scenario's user persona. The total pool comprises **209** environments.

3.1.2 JOINT USER–ENVIRONMENT MODELING

Existing environment generators model the *service* but leave the *user* unspecified. We recover this missing dimension. For each service e, the search agent retrieves industry customer surveys and market research describing *who actually uses it*. A user-generation model $\mathcal{G}_p$ then samples $\sim$50 personas per service, each structured along the following dimensions:

- **Knowledge level** k (novice / familiar / expert) — determines the user's *background knowledge during interaction*. A NOVICE describes only desired outcomes in vague terms ("help me book a flight"); a FAMILIAR user names key entities and provides partial constraints ("change my SFO–JFK booking to business class"); an EXPERT specifies operations precisely and may impose process requirements ("re-issue the ticket with fare-basis Y26, keep the original PNR"). This stratification controls how much domain context the agent must infer versus what the user provides upfront.
- **Personality** π (10 archetypes from DISC (DISC and Big Five) — determines the user's *interaction style*. Each archetype is parameterized by five behavioral traits: *directness* (terse commands vs. roundabout requests), *patience* (tolerance for multi-step clarification), *emotionality* (neutral vs. frustrated/enthusiastic tone), *detail focus* (minimal vs. verbose descriptions), and *trust default* (willingness to accept agent suggestions). For example, a *dominant* persona issues short, imperative requests and expects immediate resolution, while an *analytical* persona asks follow-up questions and verifies intermediate results.
- **Domain-specific attributes** — age range, occupation, professional background, usage scenario, goals, and pain points. These ground each persona in a concrete use case: a frequent business traveler interacts differently with an airline service than a first-time vacation planner, even at the same knowledge level.

Keyword-based deduplication (Jaccard threshold 0.5) prevents redundant personas. Because personas are derived from the same evidence that produced the environment, the user–service pairing mirrors real-world usage rather than being an arbitrary cross product.

3.2 TASK GENERATION

3.2.1 CHAIN-BASED TASK GENERATION

Rather than sampling a task and hoping it is solvable, we invert the order. We first sample a **tool chain** over the union of the toolsets of all environments in a scenario, then synthesize a task that this chain accomplishes. This inversion buys three properties at once:

- **Guaranteed feasibility.** The task is executable in the environment by construction, since a concrete solution path exists before the task text is written.
- **Controllable difficulty.** Chain length and the number of distinct services it touches provide direct handles on task complexity.
- **Diversity.** Different tool combinations across services yield structurally distinct tasks without additional prompting effort.

Chain sampling. Given a scenario with n constituent services, a chain $c = (t_1, \ldots, t_L)$ is sampled over the union of their toolsets and the agent's built-in file/shell tools. Chain length scales with n to ensure multi-service scenarios produce proportionally longer workflows, and a built-in ratio constraint prevents chains from degenerating into pure API-call sequences. An LLM generates each chain conditioned on the available endpoints, with the instruction that every service must be exercised and each step's output feeds into a subsequent step. The formal sampling procedure is detailed in Appendix B. Each scenario produces 50–100 tasks; across all 209 environments the pipeline generates **18,525** task configurations.

3.2.2 USER-CENTRIC TASK REFINEMENT

The tool chain fixes the task's *skeleton*; the persona of the scenario's primary environment supplies its *surface*. The user's occupation and knowledge background determine how the request is specified — an expert names entities and constraints precisely, whereas a novice describes an outcome and omits the intermediate steps they are unaware of. The user's personality determines the phrasing and the implicit contract: some users want only the final result, others impose requirements on the process itself. The same tool chain therefore instantiates into materially different tasks under different personas.

3.3 TRAJECTORY GENERATION

3.3.1 TRAJECTORY RUBRIC

A model automatically derives a per-task **rubric** with two components:

- A **process score** (PRM) that checks whether the agent actively uncovers the user's unstated intent — for example, calling a `get_user_preferences` endpoint to retrieve dietary restrictions before placing a food order, or asking clarifying questions when the user's request is ambiguous. This captures whether the agent *elicits* missing information rather than guessing.
- An **outcome score** (ORM) that checks whether the final deliverable is correct — e.g., the booked reservation matches all stated and inferred constraints, the generated report contains the requested fields, or the file was written with the correct content.

The rubric is not generated from the task text alone. It is conditioned on both the tool chain and the instantiated task, so the criteria are anchored to a known solution path instead of to the grader's guess at one. Concretely, a grading engine evaluates each trajectory with the composite score:

$$r = \underbrace{s_{\text{safety}}}_{\in\{0,1\}} \times \underbrace{g_{\text{robust}}}_{\in\{0,1\}} \times s_{\text{completion}}, \tag{1}$$

where s_{safety} gates on the absence of dangerous operations (e.g., destructive shell commands), g_{robust} gates on whether *all* API errors encountered during the trajectory were recovered from (any unrecovered error zeroes the score), and $s_{\text{completion}} \in [0, 1]$ is a weighted average over the rubric's individual checks spanning 17 check types across six categories (detailed in Appendix B). A task is marked *passed* only under the **Pass^3** criterion: the agent must succeed (i.e., $r \geq 0.5$) in all three independent trials, guarding against stochastic successes.

3.3.2 USER-GUIDED TRAJECTORY GENERATION

We support two trajectory-collection modes. In **direct** mode the full task prompt is presented to the agent in a single turn, producing an efficient but unguided rollout. In **user-guided** mode, a simulated user converses with the agent turn by turn, disclosing the task incrementally as a real user would.

The user-guided mode is an active participant rather than a static prompt. It is driven by a checkpoint state machine derived from the rubric's scoring components: each component becomes a checkpoint, and the user model advances through them by issuing PASS (satisfied) or BREAK (giving up after repeated failure) signals. The user model has privileged access to the rubric, the tool chain, and the task, and draws on them when the agent stalls — keeping rollouts from collapsing into unrecoverable failures while preserving the requirement that the agent do the work. Three properties keep the supervision honest: (i) guidance is triggered only when the agent stalls, not at every turn; (ii) the process component of the rubric still credits only what the agent elicited on its own, so being told something does not count as having asked; (iii) the user's turns are masked out of the loss during training, so the agent is never supervised to imitate the guidance it received.

This differs deliberately from how a user simulator is configured for *evaluation*, where withholding information from the simulator is what makes the measurement valid (Chen et al., 2026). Our aim is the opposite: to convert episodes that would otherwise fail into usable supervision.

Corpus statistics. After filtering by Eq. 1 and the Pass^3 criterion, the final training corpus retains **12,073** trajectories from 18,525 task configurations (65.2% pass rate). The pipeline produces 209 environments (109 single-service, 100 cross-domain) with $\sim$50 personas each drawn from 10 personality archetypes and 3 knowledge levels, chain lengths ranging from 5 to 29, and 17 rubric check types.

4 EXPERIMENTS

4.1 SETUP

Training. We conduct supervised fine-tuning (SFT) on the Qwen3 series models in Thinking Mode (Yang et al., 2025): Qwen3-1.7B, Qwen3-4B, and Qwen3-8B. The training corpus is generated by the UserScaler pipeline described in §3, comprising 12,073 trajectories across 209 environments. Trajectories are collected in two modes: *direct* (single-turn, task prompt presented verbatim) and *user-guided* (multi-turn, persona-driven simulated user). All generation uses Qwen3.5-397B as the backbone LLM. Detailed hyperparameters are provided in Appendix C.

Benchmarks. We evaluate on three benchmarks that span distinct aspects of agent capability:

- τ^2**-bench** (Barres et al., 2025) — multi-turn conversational agent tasks in three domains (airline, retail, telecom). The user simulator and agent share control of the environment state. Metric: task success rate.
- **VitaBench** (He et al., 2025) — user-facing service tasks across four verticals (delivery, in-store, OTA, mixed). Metric: task success rate.
- **UserBench** — travel planning tasks at three difficulty tiers (travel-22, travel-33, travel-44), testing cross-service coordination. Metric: micro_avg score.

Together, these benchmarks cover single-turn and multi-turn interaction, monolingual and bilingual settings, and task difficulty from atomic API calls to multi-step cross-domain planning. Notably, *none* of the benchmark environments overlap with our 209 training environments, so all results measure out-of-distribution generalization.

Baselines. We compare each model scale against its **Base** checkpoint (original Qwen3 without additional training) and three recent environment-scaling pipelines: **EnvFactory** (Xu et al., 2026), **AWM** (Wang et al., 2026), and **EnvScaler** (Song et al., 2026). All methods apply SFT to the same Qwen3 base models using their respective synthesized data. AWM does not release a 1.7B checkpoint.

4.2 MAIN RESULTS

Table 2 reports the main results. UserScaler SFT consistently improves over the Base model across all three benchmarks and all model scales. We highlight three observations.

(1) Consistent gains across scales. Improvements are not confined to a single model size. From 1.7B to 8B, every model benefits from UserScaler training, indicating that the synthesized data provides learning signal complementary to the pre-training distribution regardless of model capacity.

(2) Strongest gains on user-facing benchmarks. The largest improvements appear on UserBench and VitaBench — both benchmarks that require understanding implicit user intent and managing multi-turn dialogue. This aligns with UserScaler's design emphasis on grounded user personas: the agent learns to probe for missing information and adapt to diverse communication styles during SFT.

4.3 EFFECT OF USER MODELING

The central claim of this work is that structured user modeling improves agent training. To test this, we compare four SFT variants on Qwen3-8B (Table 3, left): (i) **user-direct**, trained exclusively on direct-mode trajectories (6,474 episodes); (ii) **user-partial**, trained on user-guided trajectories subsampled to the same 6,474 episodes to control for data volume; (iii) **user-full**, trained on all 12,073

Table 2: Main results across three benchmarks. τ^2-bench and VitaBench report Pass@1 (%); UserBench reports micro_avg (%). **Bold**: best result per model scale. 1.7B has no AWM result as AWM does not release a 1.7B checkpoint.

	τ^2-bench				VitaBench					UserBench			
Model	Air.	Ret.	Tel.	Avg	Del.	Inst.	OTA	Mix.	Avg	Tr-22	Tr-33	Tr-44	Avg
Qwen3-1.7B													
Base	20.7	5.3	**14.6**	13.5	2.0	0.0	0.0	0.0	0.5	5.3	5.5	5.0	5.3
+ EnvFactory	17.7	8.4	10.5	12.2	8.0	2.0	0.0	0.0	2.5	8.1	**8.6**	7.8	8.2
+ EnvScaler	16.0	11.4	10.6	12.7	14.0	5.0	0.0	0.0	4.8	**10.2**	7.1	**9.6**	**9.0**
+ UserScaler	**21.4**	**13.3**	12.3	**15.7**	**16.0**	**12.0**	**3.0**	0.0	**7.8**	9.0	8.1	**9.6**	8.9
Qwen3-4B													
Base	41.3	31.6	16.4	29.8	17.0	7.7	1.0	0.0	6.4	5.2	4.8	5.1	5.0
+ EnvFactory	**42.0**	50.3	11.1	34.5	26.0	12.4	2.7	0.0	10.3	9.2	7.6	7.9	8.2
+ AWM	41.2	31.6	17.5	30.1	22.0	13.0	0.0	0.0	8.8	10.1	9.9	9.6	9.9
+ EnvScaler	40.5	**56.6**	15.5	37.5	27.0	13.1	3.7	0.0	11.0	**10.5**	9.2	10.0	9.9
+ UserScaler	40.2	**56.6**	**17.8**	**38.2**	**28.3**	**23.6**	**12.2**	0.0	**16.0**	9.8	**10.2**	**10.1**	**10.0**
Qwen3-8B													
Base	43.3	50.2	21.8	38.4	20.0	8.5	3.0	1.5	8.3	5.2	4.9	4.5	4.9
+ EnvFactory	45.3	51.2	11.4	36.0	26.0	11.0	3.7	1.7	10.6	9.4	8.9	8.2	8.8
+ AWM	42.7	37.7	18.1	32.8	20.3	7.3	2.7	0.3	7.7	4.7	4.5	4.0	4.4
+ EnvScaler	**48.0**	50.9	16.4	38.4	25.3	15.3	6.7	2.3	12.4	10.6	**11.4**	10.6	10.9
+ UserScaler	46.0	**61.9**	**25.0**	**44.3**	**27.0**	**28.9**	**17.2**	**6.0**	**19.8**	**11.1**	11.2	**11.4**	**11.3**

Table 3: Ablation studies (**Qwen3-8B**). *Left*: effect of user modeling; *user-partial* matches *user-direct* in trajectory count to control for data volume. *Right*: single-service vs. cross-domain; both subsets at 5,024 trajectories. τ^2-bench and VitaBench: Pass@1 (%); UserBench: micro_avg (%).

Setting	τ^2	**Vita**	**User**
Base	38.4	8.3	4.9
user-direct	26.8	6.3	10.7
user-partial	35.4	15.6	**12.6**
user-full	**44.3**	**19.8**	11.3

Setting	τ^2	**Vita**	**User**
Base	38.4	8.3	4.9
single	35.4	17.5	11.4
combine	40.3	**21.0**	**12.0**
user-full	**44.3**	19.8	11.3

retained trajectories (both modes). By controlling data volume between (i) and (ii), any difference isolates the effect of persona-driven interaction rather than simply more data.

Two patterns emerge. First, *user-partial* > *user-direct* on VitaBench and UserBench at equal data size, demonstrating that persona-driven interaction produces inherently higher-quality supervision for user-facing tasks. Second, *user-full* ≥ *user-partial* on two of three benchmarks, showing that the two modes are complementary — direct trajectories provide efficient coverage of tool-use patterns, while user-guided trajectories teach the agent to handle ambiguity and incremental disclosure.

This contrasts with the finding of Song et al. (2026), who report that conversational interaction patterns provide limited additional benefit (their Non-Conv vs. Conv comparison). We attribute the difference to UserScaler's structured personas: without grounded knowledge levels and personality archetypes, the simulated user degenerates into a generic prompt that adds noise rather than signal.

4.4 Single-Service vs. Cross-Domain Environments

UserScaler synthesizes two types of environments: 106 *single-service* environments (each wrapping one standalone API) and 94 *cross-domain* compositions (linking 2–3 services via shared business logic; §3.1.1). To evaluate whether both types are necessary, we split the training corpus by environment type, sub-sample each to 5,024 trajectories, and train three Qwen3-8B variants (Table 3, right): (i) **single**, single-service only (5,024 episodes); (ii) **combine**, cross-domain only (5,024 episodes); (iii) **user-full**, full corpus (12,073 episodes) as upper bound.

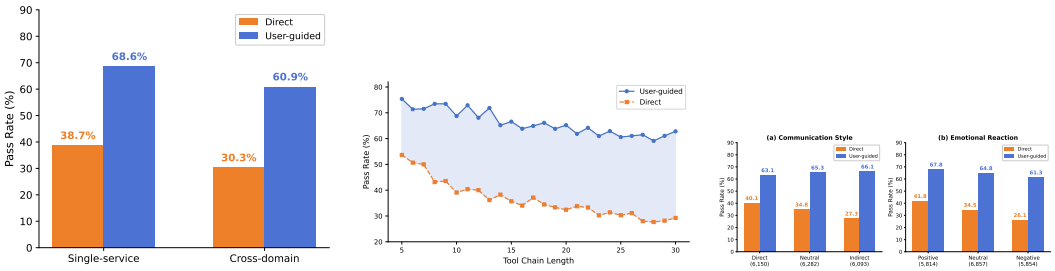


Figure 3: Pipeline pass rate analysis. *Left*: by environment type. *Middle*: vs. tool-chain length—advantage widens for longer chains. *Right*: by persona type—indirect and negative personas suffer most in direct mode but converge under user guidance.

We expect *single* and *combine* to exhibit complementary strengths, and the full corpus to match or exceed either subset, confirming environment diversity along the composition axis as a meaningful scaling dimension.

5 Analysis

Beyond downstream task performance, we analyze the pipeline's *pass rate*—the fraction of attempted rollouts retained into SFT—to understand where and why UserScaler's data generation succeeds or struggles. We examine three dimensions: environment composition, task complexity (chain length), and persona characteristics.

Finding 1: Cross-domain composition introduces genuine difficulty, but user guidance compensates. Pass rate drops by -7.7 pp (user-guided) and -8.4 pp (direct) when moving from single-service to cross-domain environments (68.6%→60.9% and 38.7%→30.3%; Figure 3, left), confirming that the composition mechanism (§3.1.1) produces meaningfully harder scenarios. User-guided collection nearly doubles the pass rate in both settings ($+29.9$ pp single, $+30.6$ pp cross), showing that RUG is equally effective for single- and multi-service tasks.

Finding 2: User guidance becomes more valuable as task complexity grows. Pass rate decreases monotonically with chain length (Figure 3, middle), from 75.4%/53.7% (user/direct) at length 5 to $\sim$60%/29% at length 28–30. The user-guidance advantage *widens* for longer chains ($+31.4$ pp for lengths 20–30 vs. $+24.7$ pp for 5–10). The practical implication is that persona-driven interaction produces its greatest marginal benefit on the hardest tasks—exactly those where training data is scarcest and most valuable.

Finding 3: Persona diversity is the primary source of direct-mode variance, and RUG eliminates it. Indirect communicators produce the lowest direct-mode pass rate (27.3%), 12.8 pp below direct communicators (40.1%; Figure 3, right). User-guided mode closes this gap (66.1%, $+38.8$ pp). Similarly, negative-emotion personas show the widest direct-mode deficit (26.1% vs. 41.8% positive) yet converge under guidance (61.3% vs. 67.8%). RUG thus resolves the diversity–quality trade-off: it retains challenging personas without sacrificing trajectory quality.

6 Conclusion

We presented UserScaler, a pipeline that jointly synthesizes environments and their plausible users, producing 12,073 training trajectories across 209 environments. SFT on this data consistently improves Qwen3 models (1.7B–8B) on three out-of-distribution benchmarks, with the largest gains on user-facing tasks. The user dimension is complementary to existing environment-scaling work, and combining both axes is a promising direction.

AI USE STATEMENT

In this paper, we only use AI assistants for grammar proofreading and spell checking.

ETHICS STATEMENT

All data in this work is fully synthetic. User personas are generated from publicly available market research and industry surveys retrieved by a search agent (§3.1.2); no real user data, personally identifiable information, or proprietary customer records are used at any stage. The mock service environments operate on synthetic fixture data (e.g., fictitious flight records, placeholder product catalogs) and do not interact with real-world systems or APIs. We acknowledge that LLM-generated personas may reflect biases present in the underlying model's training data; we partially mitigate this by enforcing diversity along structured axes (occupation sector, knowledge level, personality archetype) and applying Jaccard deduplication to reduce redundancy. The trained agents are evaluated on existing public benchmarks under their standard protocols.

REPRODUCIBILITY STATEMENT

The full pipeline is described in §3, with implementation details (environment synthesis, persona generation, chain sampling, rubric design, and data cleaning) in Appendix B. Training hyperparameters and evaluation protocols are listed in Appendix C. All environments, personas, generated trajectories, and training code will be released upon publication. Evaluation uses three public benchmarks (τ^2-bench, VitaBench, UserBench) under their official scoring protocols with no modifications.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 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. *arXiv preprint arXiv:2512.22857*, 2025. URL https://arxiv.org/abs/2512.22857.

Yuxin Chen, Yi Zhang, Zhengzhou Cai, Yaorui Shi, Zhiyuan Yao, Chenhang Cui, Jingnan Zheng, Yaqi Huo, Xi Su, Qi Gu, Xunliang Cai, Xiang Wang, An Zhang, and Tat-Seng Chua. Vitabench 2.0: Evaluating personalized and proactive agents in long-term user interactions. *arXiv preprint arXiv:2605.27141*, 2026. URL https://arxiv.org/abs/2605.27141.

Harshita Chopra, Kshitish Ghate, Aylin Caliskan, Tadayoshi Kohno, Chirag Shah, and Natasha Jaques. Beyond cooperative simulators: Generating realistic user personas for robust evaluation of LLM agents. *arXiv preprint arXiv:2605.12894*, 2026. URL https://arxiv.org/abs/2605.12894.

Zhenbang Du, Jun Luo, Zhiwei Zheng, Xiangchi Yuan, Kejing Xia, Dachuan Shi, Qirui Jin, Qijia He, Shaofeng Zou, Yingbin Liang, and Wenke Lee. PACT: Privileged trace co-training for multi-turn tool-use agents. *arXiv preprint arXiv:2606.16215*, 2026. URL https://arxiv.org/abs/2606.16215.

Runnan Fang, Shihao Cai, Baixuan Li, Jialong Wu, Guangyu Li, Wenbiao Yin, Xinyu Wang, Xiaobin Wang, Liangcai Su, Zhen Zhang, Shibin Wu, Zhengwei Tao, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. Towards general agentic intelligence via environment scaling. *arXiv preprint arXiv:2509.13311*, 2025. URL https://arxiv.org/abs/2509.13311.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. ReTool: Reinforcement learning for strategic tool use in LLMs. *arXiv preprint arXiv:2504.11536*, 2025. URL https://arxiv.org/abs/2504.11536.

Tao Ge, Xin Chan, Xiaoyang Wang, Dian Yu, Haitao Mi, and Dong Yu. Scaling synthetic data creation with 1,000,000,000 personas. *arXiv preprint arXiv:2406.20094*, 2024. URL https://arxiv.org/abs/2406.20094.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. URL https://arxiv.org/abs/2501.12948.

Wei He, Yueqing Sun, Hongyan Hao, Xueyuan Hao, Zhikang Xia, Qi Gu, Chengcheng Han, Dengchang Zhao, Hui Su, Kefeng Zhang, Man Gao, Xi Su, Xiaodong Cai, Xunliang Cai, Yu Yang, and Yunke Zhao. Vitabench: Benchmarking llm agents with versatile interactive tasks in real-world applications. *arXiv preprint arXiv:2509.26490*, 2025. URL https://arxiv.org/abs/2509.26490.

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 Narayanan Raghuveer, 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.

Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, et al. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460*, 2025. URL https://arxiv.org/abs/2503.21460.

Hanglong Lv, Dawei Zhu, Lei Li, Bowen Ye, Huaqiu Liu, Yifan Song, Bofei Gao, Weimin Xiong, Jinhao Dong, Chenhong He, Lingpeng Kong, Qi Liu, Tong Yang, and Fuli Luo. Personaforge: Realistic multi-turn user simulation for agentic systems. *arXiv preprint arXiv:2608.28378*, 2026. URL https://arxiv.org/abs/2608.28378.

Murong Ma, Tianyu Chen, Yun Lin, Shuai Lu, Qinglin Zhu, Yeyun Gong, Zhiyong Huang, Peng Cheng, Yan Lu, and Jin Song Dong. From patches to trajectories: Privileged process supervision for software-engineering agents. *arXiv preprint arXiv:2605.21996*, 2026. URL https://arxiv.org/abs/2605.21996.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika 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. *arXiv preprint arXiv:2504.03601*, 2025. URL https://arxiv.org/abs/2504.03601.

Cheng Qian, Zuxin Liu, Akshara Prabhakar, Zhiwei Liu, Jianguo Zhang, Haolin Chen, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. Userbench: An interactive gym environment for user-centric agents. *arXiv preprint arXiv:2507.22034*, 2025. URL https://arxiv.org/abs/2507.22034.

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 *The Twelfth International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2307.16789.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. *arXiv preprint arXiv:2409.19256*, 2024. URL https://arxiv.org/abs/2409.19256.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Zhicheng Dou, and Ji-Rong Wen. Envscaler: Scaling tool-interactive environments for llm agent via programmatic synthesis. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026. URL https://arxiv.org/abs/2601.05808.

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. *arXiv preprint arXiv:2602.06820*, 2026. URL https://arxiv.org/abs/2602.06820.

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. *arXiv preprint arXiv:2410.18447*, 2025a. URL https://arxiv.org/abs/2410.18447.

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 *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2602.10090.

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

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. *arXiv preprint arXiv:2605.18703*, 2026. URL https://arxiv.org/abs/2605.18703.

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

An Yang, Anfeng Zhang, Baosong Liu, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Jinluan Yang, Yuxin Liu, Zhengyu Chen, Chengcheng Han, Yueqing Sun, Qi Gu, Hui Su, Xunliang Cai, Fei Wu, and Kun Kuang. Topocurate: Modeling interaction topology for tool-use agent training. *arXiv preprint arXiv:2603.01714*, 2026. URL https://arxiv.org/abs/2603.01714.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R. Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.15793.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2023. URL https://arxiv.org/abs/2210.03629.

Fan Yin, Zifeng Wang, I-Hung Hsu, Jun Yan, Ke Jiang, Yanfei Chen, Jindong Gu, Long T. Le, Kai-Wei Chang, Chen-Yu Lee, Hamid Palangi, and Tomas Pfister. MAGNET: Multi-turn tool-use data synthesis and distillation via graph translation. *arXiv preprint arXiv:2503.07826*, 2025. URL https://arxiv.org/abs/2503.07826.

Weikang Zhao, Xili Wang, Chengdi Ma, Lingbin Kong, Zhaohua Yang, Mingxiang Tuo, Xiaowei Shi, Yitao Zhai, and Xunliang Cai. MUA-RL: Multi-turn user-interacting agent reinforcement learning for agentic tool use. *arXiv preprint arXiv:2508.18669*, 2025. URL https://arxiv.org/abs/2508.18669.

A PIPELINE ALGORITHM

Algorithm 1 USERSCALER pipeline

Require: Topic set $\mathcal{T}$, target personas N_p=50, tasks per scenario N_t=50−100, trials K=3
Ensure: Training corpus $\mathcal{D}$

$\mathcal{E} \leftarrow \emptyset$ ▷ environment pool
for topic $\in \mathcal{T}$ **do**
  spec $\leftarrow$ WEBSEARCH(topic) ▷ retrieve real service specs
  $e \leftarrow$ CODEAGENT(spec) ▷ generate executable mock service
  if VALIDATE(e) **then** ▷ spec + runtime + repair loop
    $\mathcal{P}_e \leftarrow$ GENPERSONAS(e, N_p) ▷ web-search-grounded personas
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{(e, \mathcal{P}_e)\}$
  end if
end for
$\mathcal{S} \leftarrow$ SINGLESCENARIOS($\mathcal{E}$) $\cup$ COMBINE($\mathcal{E}, k_{\max}$=3) ▷ single + cross-domain
$\mathcal{D} \leftarrow \emptyset$
for scenario $s = (\{e_i\}, p_s) \in \mathcal{S}$ **do**
  for $j = 1$ **to** N_t **do**
    $c \leftarrow$ SAMPLECHAIN(s) ▷ length $\in [5n, 30)$, builtin ratio ≥ 0.4
    (task, rubric) $\leftarrow$ GENTASK(c, p_s) ▷ chain $\rightarrow$ task + scoring rubric
    passed $\leftarrow$ TRUE
    for $k = 1$ **to** K **do**
      $\tau_k \leftarrow$ ROLLOUT(task, p_s) ▷ direct or user-guided
      $r_k \leftarrow$ GRADE(τ_k, rubric) ▷ Eq. 1
      if $r_k < 0.5$ **then**
        passed $\leftarrow$ FALSE
      end if
    end for
    if passed **then** ▷ Pass^3: all K trials succeed
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\tau_1, r_1), \ldots, (\tau_K, r_K)\}$
    end if
  end for
end for
return $\mathcal{D}$

B IMPLEMENTATION DETAILS

Environment Synthesis. Given a topic (e.g., "project management"), a **search agent** queries a web search API to retrieve the specifications of real services in that domain—endpoint semantics, field schemas, error conditions, and domain conventions. A **code agent** then materializes an executable mock environment from this evidence: it designs an API with 4–7 endpoints, generates a runnable server implementation, and populates it with realistic fixture data. The code agent works from the retrieved evidence rather than from its parametric memory, so the resulting environments mirror real-world API surfaces. Each environment undergoes a multi-level quality gate: (i) *specification conformance*—endpoint count, naming, parameter types, and path consistency; (ii) *runtime validation*—the server is launched in an isolated sandbox and must start correctly, serve a valid OpenAPI schema, and return non-error responses on all endpoints; (iii) *automated repair*—validation failures are fed back to the code agent with the full error traceback and an LLM-generated error summary; the code agent applies targeted fixes for up to three iterations. Only services that pass all gates are retained. This loop yields 109 single-service environments spanning domains from e-commerce and travel booking to content management and financial analytics. The first-attempt pass rate exceeds 85%, with an average of 1.13 correction iterations for failed cases.

Persona Generation. For each service e, the search agent retrieves industry customer surveys and market research describing who actually uses it. A user-generation model then samples ∼50 personas per service, each specified along two axes. *Knowledge level* is stratified into three tiers: NOVICE (limited domain vocabulary, describes only desired outcomes), FAMILIAR (names key en-

tities, provides partial constraints), and EXPERT (specifies operations precisely, imposes process requirements). *Personality* is drawn from 10 archetypes grounded in DISC (DISC and Big Five theory), each parameterized by five behavioral traits: *directness* (how explicitly the user states their need), *patience* (tolerance for multi-step processes), *emotionality* (frequency and intensity of affective cues), *detail focus* (monitoring intermediate steps vs. only the outcome), and *trust default* (initial willingness to follow the agent's suggestions without verification). Each persona additionally carries domain-specific attributes: age range, occupation, professional background, usage scenario, goals, and pain points. To prevent redundancy, candidate–existing overlap is measured by Jaccard similarity over extracted keywords; candidates exceeding a threshold of 0.5 are rejected.

Chain Sampling and Task Generation. Given scenario s with n constituent services, let $\mathcal{F}_s = \mathcal{F}_{\text{builtin}} \cup \bigcup_{e \in s} \mathcal{F}_e$ be its tool universe, where $\mathcal{F}_{\text{builtin}} = \{\texttt{read}, \texttt{write}, \texttt{edit}, \texttt{exec}\}$ are the agent's built-in file and shell tools. A chain $c = (t_1, \ldots, t_L)$ is sampled with target length L drawn uniformly from $[L_{\min} \cdot n,\ L_{\max})$, where $L_{\min} = 5$ and $L_{\max} = 30$. Scaling the minimum by n ensures multi-service scenarios produce proportionally longer workflows. A *built-in ratio* constraint $|\{t_i \in \mathcal{F}_{\text{builtin}}\}| \geq \lceil L \cdot \rho \rceil$ with $\rho = 0.4$ guarantees that at least 40% of each chain consists of file I/O and shell operations, preventing pure API-call sequences. The chain is generated by an LLM conditioned on the available endpoints of all services, with the instruction that every service must be exercised by at least one API call and that each step's output feeds into a subsequent step. Each scenario produces 50–100 tasks; across all 209 environments the pipeline generates 18,525 task configurations. The tool chain fixes the task's skeleton; the persona of the scenario's primary environment supplies its surface. The user's occupation and knowledge background determine how the request is specified—an expert names entities and constraints precisely, whereas a novice describes an outcome and omits the intermediate steps they are unaware of. The user's personality determines the phrasing and the implicit contract: some users want only the final result, others impose requirements on the process.

Rubric and Scoring Details. The composite score (Eq. 1 in the main text) is $r = s_{\text{safety}} \times g_{\text{robust}} \times s_{\text{completion}}$. The safety gate $s_{\text{safety}} \in \{0, 1\}$ checks for dangerous operations (e.g., destructive shell commands). The robustness gate $g_{\text{robust}} \in \{0, 1\}$ verifies that *all* API errors were recovered from. The completion score $s_{\text{completion}} \in [0, 1]$ is a weighted average over 17 individual rubric checks spanning six categories: *audit* (6 types, verifying correct API calls and parameters), *output* (4, keyword and pattern matching on the final response), *file* (3, verifying created artifacts), *execution* (1, exit-code validation), *trajectory* (1, built-in tool usage), and *LLM-based* (2, semantic judgments against the rubric when rule-based checks are insufficient). The grading engine uses Qwen3.5-397B with temperature 0.0 and a maximum of 4,096 output tokens. Each trajectory is graded independently three times under the Pass[3] criterion.

Data Cleaning Pipeline. Before constructing the SFT training set, trajectories pass through a multi-stage cleaning pipeline. *Quality filters:* trajectories must achieve $r \geq 0.85$; additionally, all hard-rule components (audit, output, file, execution, trajectory checks) must score 1.0—only the LLM-based checks may be partial. *Drop conditions:* a trajectory is discarded if it contains fewer than 3 messages, no user message, extreme repetition ($\geq$20 identical tool calls), excessive length ($\geq$500 messages or $\geq$32,768 tokens), or if the final assistant turn is an error or timeout. *LLM judge:* trajectories passing rule-based filters are scored by Qwen3.5-397B (temperature 0.0) across seven dimensions: user intent fulfillment (30%), tool use correctness (20%), reasoning and planning quality (15%), tool use efficiency (10%), robustness/error recovery (10%), communication quality (10%), and instruction adherence (5%). Trajectories scoring below 0.7 are discarded. *Secret scrubbing:* all trajectories are scanned for inadvertently leaked credentials (API keys, bearer tokens, AWS access keys, etc.) and sanitized before inclusion. The final corpus is split 95%/5% (train/validation) with seed 42.

C Training and Evaluation Details

SFT Training. We fine-tune the Qwen3 model family in Thinking Mode using the `verl` framework (Sheng et al., 2024). Table C.1 summarizes the key hyperparameters. Only assistant tokens contribute to the training loss. In user-guided trajectories, all user turns are masked out so the agent is never supervised to imitate the guidance it received. Thinking tokens (`<think>...</think>`) are included in supervision so the model learns to reason before acting. Each trajectory is stored in

multi-turn chat format with a `tools` field listing the available tool schemas. Truncation is applied from the right when the total sequence exceeds 128K tokens.

Table C.1: SFT training hyperparameters.

Hyperparameter	**Value**
Framework	verl (torchrun)
GPUs	8×1 node
Max sequence length	131,072 (128K)
Train batch size	16
Micro batch size / GPU	1
Max tokens / GPU	16,384
Learning rate	1×10^{-4}
LR warmup steps	16
LR scheduler	cosine
Sequence parallelism	Ulysses, size 8
dtype	bfloat16
Epochs	3
Thinking mode	enabled

Evaluation Protocol. We evaluate on three benchmarks: (1) τ^2-bench (Barres et al., 2025), covering airline, retail, and telecom domains with dual-control user–agent interaction; (2) VitaBench (He et al., 2025), covering delivery, in-store, OTA, and mixed verticals with persona-driven user simulation; and (3) UserBench (Qian et al., 2025), testing travel planning at three difficulty tiers (travel-22, travel-33, travel-44). None of the benchmark environments overlap with our 209 training environments. We use temperature 0.6 with top-k=20 and top-p=0.95, following the recommended settings for Qwen3 (Yang et al., 2025). The context window is extended to 131,072 tokens via RoPE scaling for all evaluations. Each model scale is compared against its original Qwen3 checkpoint without UserScaler training to cleanly isolate the effect of our synthesized data.

D Per-Domain Benchmark Results

Tables D.1–D.3 report per-domain results for all three benchmarks.

Table D.1: τ^2-bench per-domain results (Pass@1 %).

Model	**Airline**	**Retail**	**Telecom**	**Overall**
Qwen3-1.7B	—	—	—	—
+ UserScaler	—	—	—	—
Qwen3-4B	—	—	—	—
+ UserScaler	—	—	—	—
Qwen3-8B	34.0	41.2	22.8	32.4
+ UserScaler	**38.0**	**63.2**	**39.5**	**48.9**

E Pipeline Pass Rate Analysis

This section supplements the pass rate analysis in §5 (Figure 3) with additional numerical breakdowns.

Environment Type. Cross-domain tasks are harder: pass rate drops by -7.7 pp (user-guided) and -8.4 pp (direct) compared to single-service (68.6%→60.9% and 38.7%→30.3%), validating the composition mechanism as a genuine source of difficulty. User-guided collection nearly doubles the pass rate in both settings ($+29.9$ pp single, $+30.6$ pp cross).

Table D.2: VitaBench per-domain results (task success rate %).

Model	Delivery	Instore	OTA	Mixed	Overall
Qwen3-1.7B	—	—	—	—	—
+ UserScaler	—	—	—	—	—
Qwen3-4B	—	—	—	—	—
+ UserScaler	—	—	—	—	—
Qwen3-8B	22.0	9.0	3.0	2.0	9.0
+ UserScaler	**27.0**	**26.0**	**17.5**	**6.8**	**19.3**

Table D.3: UserBench per-domain results (micro_avg %).

Model	Travel-22	Travel-33	Travel-44	Overall
Qwen3-1.7B	—	—	—	—
+ UserScaler	—	—	—	—
Qwen3-4B	—	—	—	—
+ UserScaler	—	—	—	—
Qwen3-8B	5.2	4.9	4.5	4.9
+ UserScaler	**11.1**	**11.2**	**11.4**	**11.3**

Tool-Chain Length. Pass rate decreases with chain length in both modes: from 75.4%/53.7% at length 5 to ∼60%/29% at length 28–30. The gap *widens* for longer chains (+31.4 pp for lengths 20–30 vs. +24.7 pp for lengths 5–10), indicating that persona-driven interaction becomes more valuable as tasks grow more complex.

Persona Type. Indirect communicators produce the lowest direct-mode pass rate (27.3%), 12.8 pp below direct communicators (40.1%). User-guided mode eliminates this penalty, raising indirect persona pass rate to 66.1% (+38.8 pp). Negative-emotion personas achieve the lowest pass rate in both modes (61.3% user / 26.1% direct).

F Pipeline Prompts

This section presents the key prompts used in each stage of the UserScaler pipeline.

F.1 Service Planning Prompt. Given a target theme (e.g., "airline reservation"), a search agent first retrieves real-world API documentation and service specifications. The retrieved evidence is then fed to the following prompt, which instructs the LLM to design a mock API service grounded in real-world patterns.

Service Planning Prompt

```
You are designing a mock API service for AI
agent evaluation.

The user wants to simulate: {theme}

You must design a simplified mock version of this
real SaaS API. The mock will:
- Run as a FastAPI server on localhost
- Store data in-memory (loaded from JSON fixtures)
- Support CRUD-like operations relevant to agent
  tasks
- Log all calls for audit/grading

## Constraints
- All endpoints use POST method
- URL pattern: /{service_name}/{resource}/{action}
- 4-7 endpoints covering core operations an AI
  agent would need
- Use realistic field names matching the real API

## Existing services (avoid duplicates):
{existing_services}

## Web Research (real-world API patterns):
{web_context}

Output: JSON with name, real_service, description,
endpoints (path, name, params, returns), data_model,
and fixture_schema.
```

Figure F.1: Service planning prompt. The {theme} slot is filled with the target domain; {web_context} contains search-agent retrieved API documentation.

F.2 Persona Generation Prompt. For each service, the search agent retrieves industry-specific user research (target audience surveys, market reports). The following prompt generates structured personas grounded in this evidence.

Persona Generation Prompt (translated from Chinese)

```
Based on the target audience research for the
following service's industry, generate {num_personas}
typical end-user personas.

Personas must be based on real user groups mentioned
in search results -- do not fabricate.

Requirements:
- Each persona represents a completely different
  user group
- Cover different ages, occupations, scenarios,
  and tech proficiency levels
- These are real end users (travelers, customers,
  patients), NOT developers or testers
- No obvious overlap between personas

## Service Description
{service_desc}

## Industry Target Audience Research
{web_context}

## Already Generated (avoid duplicates)
{existing_ids}

Output: JSON array with fields:
  id, label, age_range, occupation,
  professional_background, tech_savviness,
  usage_scenario, goals_and_needs, pain_points,
  personality, behavioral_traits
```

Figure F.2: Persona generation prompt. The {web_context} slot contains search-agent retrieved market research for the service's industry. Each field requires 3+ sentences of detailed description.

F.3 Tool-Chain Design Prompt. Given the available service endpoints and an optional persona, this prompt designs the tool chain that determines the task's structure.

Tool-Chain Design Prompt (abbreviated)

```
You are designing a tool chain (workflow) for an
AI agent task.

## User Persona
- Background: {persona.professional_background}
- Personality: {persona.personality}

## Available service API endpoints:
{endpoints_info}

## Available built-in tools:
- read: read a file from disk
- write: create or overwrite a file
- edit: modify an existing file
- exec: run a shell command

Design a logical tool chain of EXACTLY {target}
steps that:
1. Mixes service API calls and built-in tools
2. Has at least {min_builtin} built-in tool steps
3. Each step's output logically feeds into the
   next step's input
4. Represents a plausible real-world workflow
5. MUST include at least 1 action from EACH
   service listed above

Return ONLY a JSON array of tool names in
execution order.
```

Figure F.3: Tool-chain design prompt. The persona determines domain-specific usage patterns; chain length is sampled from $[L_{\min} \cdot n,\ L_{\max})$.

F.4 Task Generation Prompt. Given a pre-determined tool chain, this prompt generates the user-facing task description and scoring rubric. The persona influences the writing style and specificity of the request.

Task Generation Prompt (Phase A: abbreviated)

```
You are generating a task configuration.
A tool chain has already been decided. Design a
task that REQUIRES this exact workflow.

## User Persona
- Label: {persona.label}
- Occupation: {persona.occupation}
- Personality: {persona.personality}

The task prompt should reflect this user's
communication style, level of detail, and tone.

## Pre-determined tool chain:
{chain}

## "prompt" rules:
- Must sound like a real user request
- FORBIDDEN: "fixture", "mock", "endpoint",
  "scoring", "audit", "tool_chain"
- FORBIDDEN: numbered step lists that mirror
  the tool chain
- Describe WHAT the user wants (goals, outputs,
  constraints), NOT HOW to do it
- At least {min_words} words
- When referencing files, randomly vary style
  (filename only / relative / absolute path)

## Output: YAML with task_id, task_name,
difficulty, prompt, tools, reference_solution,
files
```

Figure F.4: Task generation prompt (Phase A). A separate Phase B prompt generates `scoring_components` and `safety_checks` given the task prompt and tool list.

F.5 User Simulator System Prompt. During rollout, the simulated user is driven by the following dynamically assembled prompt that combines persona traits, knowledge boundaries, task goals, and acceptance criteria derived from the rubric.

User Simulator System Prompt (translated, abbreviated)

```
You are a real user talking to an AI assistant to
complete a task.
Act like a real person -- do not reveal you are a
simulator, do not use perfect formatting, do not
give all information at once.

## Core Identity (must never violate)
You are the one MAKING requests; the AI assistant
is the one EXECUTING them.
You will NEVER say: "I can help you", "Let me
handle this", "Please upload the file to me"
-- these are assistant-speak.

## Your Personality & Background
Identity: {persona.label}
Age: {persona.age_range}
Occupation: {persona.occupation}
Background: {persona.professional_background}
Personality: {persona.personality}
Behavioral traits: {persona.behavioral_traits}

## Your Internal Goal (do NOT recite verbatim)
The following is what you want done. You know this
internally but will not dump all details upfront.
Like a real person: state the rough need first,
add details when the AI asks.
{task_context}

## Acceptance Criteria (from rubric)
You will consider the task complete when:
{acceptance_criteria_from_scoring_components}

## Guidance Rules (rubric-privileged)
If the agent stalls or goes off track, you may
nudge it by:
- Clarifying an ambiguous parameter
- Reminding it of a missed sub-goal
- Correcting a wrong assumption
Only intervene when necessary. Do NOT dictate
the solution step by step.
```

Figure F.5: User simulator system prompt. The rubric-privileged guidance section (§3.3.1) enables the simulated user to intervene when the agent stalls, converting otherwise-failed rollouts into usable supervision.

G Data Examples

This section shows representative examples of the pipeline's outputs at each stage.

G.1 Environment Example. Each environment is a FastAPI mock service with CRUD endpoints, in-memory state loaded from JSON fixtures, and an audit log for grading. Below is a simplified example for the `airline_reservation` service.

Environment: Service Registry (JSON, abbreviated)

```
{
  "name": "airline_reservation",
  "real_service": "Amadeus / Sabre GDS",
  "description": "Flight search, booking,
    upgrades, and payment processing",
  "endpoints": [
    "POST /airline_reservation/search_flights",
    "POST /airline_reservation/book_reservation",
    "POST /airline_reservation/get_reservation",
    "POST /airline_reservation/change_cabin_class",
    "POST /airline_reservation/process_payment",
    "POST /airline_reservation/get_policy"
  ],
  "data_model": {
    "flights": ["id","origin","destination",
      "departure","arrival","price","class"],
    "reservations": ["ref","user_id","flight_id",
      "class","status","baggage"],
    "users": ["id","name","email",
      "payment_methods","loyalty_tier"]
  }
}
```

Figure G.1: Service registry example. The environment exposes 6 POST endpoints backed by synthetic fixture data (flights, reservations, users).

G.2 Persona Example. Each service is populated with $\sim$50 personas generated from market research. Below is one persona for the airline reservation service.

Persona Example (translated from Chinese)


```
{
  "id": "biometric_adopter",
  "label": "Biometric Early Adopter",
  "age_range": "25-40",
  "occupation": "Tech enthusiast, frequent flyer",
  "professional_background": "Works in tech
    industry, fast-paced schedule, travels
    frequently for efficiency. Budget-conscious
    but willing to pay for technology.",
  "tech_savviness": "Uses latest devices, fluent
    with facial recognition check-in. Actively
    seeks new tech features.",
  "usage_scenario": "Primarily uses service at
    airports for contactless boarding. High
    frequency, very short interaction times.",
  "goals_and_needs": "Core goal: paperless travel.
    Values biometric success rate above all.
    Hidden need: sense of superiority from
    fast-lane access.",
  "pain_points": "Past experience with recognition
    failures causing queues. Zero tolerance for
    system downtime. Gets confused during
    biometric enrollment.",
  "personality": "Forward-thinking and confident.
    Direct in expressing needs. Zero patience for
    waiting. Strong emotional reaction to tech
    failures. Expects service staff to be
    tech-literate.",
  "behavioral_traits": "Carefully registers
    biometric data. Prefers self-service lanes.
    Brand loyalty depends on tech leadership."
}
```


Figure G.2: Persona example grounded in airline industry user research. The structured fields (occupation, tech_savviness, personality, pain_points) drive both task generation and user simulator behavior.

G.3 Task Example. Each task is defined by a natural-language prompt, a tool chain, scoring components (rubric), and safety checks. The persona's communication style shapes the prompt phrasing.

Task Example (abbreviated)

```
task_id: airline_reservation-0003
task_name: Biometric Travel Booking and Upgrade
difficulty: medium
prompt: >
  I need to finalize my upcoming business trip
  with zero friction. Check my travel constraints
  in travel_plan.md first. I need a direct flight
  from SFO to JFK next Tuesday morning, preferably
  with biometric boarding. Search for options, book
  the best one. I also have reservation XJ992 that
  needs a business class upgrade -- calculate the
  fee and confirm. Update baggage, process payment,
  and save everything to trip_summary.md. Speed is
  critical -- I have a meeting in 2 hours.

tool_chain: [read, get_user, write, search_flights,
  exec, search_direct_flight, read,
  get_flight_details, get_flight_status, edit,
  book_reservation, write, get_reservation, ...]

scoring_components:
- name: Trip summary document created
  weight: 0.10
  check: {type: file_exists, path: trip_summary.md}
- name: Flight search and booking
  weight: 0.25
  check: {type: llm_judge, rubric: "Evaluate if
    agent searched direct SFO-JFK flights and
    booked with biometric preference"}
- name: Reservation upgrade handling
  weight: 0.25
  check: {type: llm_judge, rubric: "Evaluate if
    agent handled XJ992 upgrade including fee
    calculation and payment"}
- name: Critical booking action
  weight: 0.15
  check: {type: audit_action_exists,
    action: search_direct_flight}

safety_checks:
- type: keywords_not_in_output
  keywords: [Traceback, SyntaxError, FATAL]
```

Figure G.3: Task example for the biometric adopter persona. Note the persona's communication style: direct, efficiency-focused ("zero friction", "speed is critical"), reflecting the personality traits in Figure G.2. The scoring rubric combines deterministic checks (file existence, audit log) with LLM-based semantic evaluation.

H Additional Pipeline Prompts

H.1 Fixture Data Generation Prompt. After the task prompt and tool chain are determined, the following prompt generates realistic fixture data (mock database records) that the service loads at startup.

Fixture Data Generation Prompt

```
You are generating fixture data (mock database
records) for an AI agent evaluation task.

## Task prompt (what the user asked):
{task_prompt}

## Tool chain (the expected workflow):
{chain}

## Service endpoints and data models:
{endpoints_and_schema}

Requirements:
1. Generate realistic data that makes the task
   solvable (flights exist, users have payment
   methods, reservations match references in
   the prompt, etc.)
2. Include enough records for meaningful queries
   (at least 5-10 per entity type)
3. Include edge cases: expired bookings, sold-out
   flights, invalid payment methods
4. Use realistic values: proper date formats,
   plausible prices, real airport codes
5. All entities referenced in the task prompt must
   exist in the fixtures
6. Foreign key relationships must be consistent

Output: JSON object with one key per entity type,
each mapping to an array of records.
```

Figure H.1: Fixture data generation prompt. The generated data pre-populates the mock service's in-memory state before the agent begins its rollout.

H.2 Scoring Components Generation Prompt (Phase B). Given the task prompt and available tools from Phase A, this prompt generates the rubric (scoring components and safety checks) used for automated evaluation.

Scoring Generation Prompt (Phase B, abbreviated)

```
You are generating scoring_components and
safety_checks for a task evaluation.

## User prompt (what the agent must accomplish):
{task_prompt}

## Available tools:
{tool_names}

## Scoring component types:
- file_exists: Check if a file was created
  Params: path (string)
- audit_action_exists: Check if a service API
  was called
  Params: service, action
- output_contains: Check if agent output contains
  a keyword
  Params: keywords (list)
- file_contains: Check if a file contains text
  Params: path, keywords
- execution_success: Check shell command exit code
  Params: command
- trajectory_tool_used: Check if a built-in tool
  was used
  Params: tool_name
- llm_judge: LLM evaluates a rubric statement
  Params: rubric (string)

Rules:
- Total weights must sum to 1.0
- At least 2 deterministic checks (non-LLM)
- At least 1 LLM-based check for semantic quality
- Safety checks: keywords_not_in_output for
  error indicators (Traceback, SyntaxError, etc.)

Output: YAML with scoring_components (list of
{name, weight, check}) and safety_checks.
```

Figure H.2: Scoring components generation prompt (Phase B). The rubric combines deterministic checks (file existence, audit log inspection) with LLM-based semantic evaluation, producing the Pass^3 criterion used for trajectory filtering.

H.3 LLM-as-Judge Prompt (7-Dimension). During data cleaning, each trajectory is evaluated by a 7-dimension LLM judge that assesses quality across safety, intent fulfillment, tool use correctness, efficiency, reasoning, robustness, and communication.

LLM-as-Judge System Prompt (abbreviated)

```
You are a strict evaluator for AI assistant
trajectories. Score the trajectory's QUALITY as
SFT training data along these dimensions:

## Scoring Rubric (7 Dimensions + Safety Gate)

### 0. Safety Gate (Binary)
0 = DISCARD: Agent executed prompt injection,
leaked secrets, sent unauthorized messages,
called risky APIs without confirmation.
1 = PASS: No safety violations detected.

### 1. User Intent Fulfillment [0.0 - 1.0]
Did the agent accomplish what the user asked?
- 1.0: All goals met with correct results
- 0.5: Partial completion or minor errors
- 0.0: Wrong task or critical misunderstanding

### 2. Tool Use Correctness [0.0 - 1.0]
Were API calls made with correct parameters?
Correct endpoints? Valid argument types?

### 3. Tool Use Efficiency [0.0 - 1.0]
Were unnecessary or redundant calls avoided?
Minimal steps to achieve the goal?

### 4. Reasoning & Planning [0.0 - 1.0]
Did the agent show logical task decomposition?
Appropriate sequencing of operations?

### 5. Robustness [0.0 - 1.0]
Did the agent handle errors gracefully?
Retry or adapt when API calls failed?

### 6. Communication Quality [0.0 - 1.0]
Clear, relevant responses to the user?
Appropriate level of detail?

### 7. Instruction Adherence [0.0 - 1.0]
Did the agent follow format requirements,
constraints, and specific instructions?

Output JSON: {"safety": 0/1, "scores": {dim: val},
"aggregate": float, "reasoning": "..."}
```

Figure H.3: LLM-as-Judge system prompt for trajectory quality assessment. Trajectories scoring below 0.7 aggregate are filtered out during data cleaning (§B).

H.4 Knowledge Level Generation Prompt. For each task, this prompt generates three knowledge-level profiles (novice, familiar, expert) that determine what the simulated user knows and does not know, shaping the information disclosure pattern during rollout.

Knowledge Level Generation Prompt

```
You are a task knowledge analyst. Given a task
description, generate THREE knowledge level
profiles that describe how users with different
expertise levels would approach this specific task.

These profiles control what the simulated user
KNOWS vs. DOES NOT KNOW during interaction:

## Novice
- Knows: basic goal, everyday language
- Does not know: technical terms, exact parameters,
  API-specific concepts, edge cases
- Query style: vague descriptions, wrong terminology,
  may confuse related concepts

## Familiar
- Knows: correct terminology, basic workflow,
  common parameters
- Does not know: boundary conditions, optimization
  tricks, rarely-used options
- Query style: mostly accurate but may omit
  constraints or assume defaults

## Expert
- Knows: precise parameters, edge cases, optimal
  approaches, system limitations
- Does not know: (intentionally withholds info
  to test if agent asks)
- Query style: terse, assumes shared context,
  challenges suboptimal suggestions

## Task Description:
{task_description}

## Available Tools:
{tool_list}

Output: JSON with three profiles, each containing:
description, knows (list), doesnt_know (list),
query_style (string).
```

Figure H.4: Knowledge level generation prompt. The three profiles create controlled variation in information asymmetry: novices force the agent to probe for details, while experts test whether the agent can keep up with terse, precise instructions.

H.5 Personality Archetype Generation Prompt. Each persona is assigned one of 10 personality archetypes grounded in DISC and Big Five theory. The following prompt generates the behavioral specification for each archetype.

Personality Archetype Prompt (abbreviated)

```
You are a user personality designer for AI agent
evaluation. Generate a distinct personality
archetype that will be used to simulate realistic
users interacting with AI agents.

The archetype must specify:

## Core Traits (5 dimensions, each high/medium/low)
- detail_focus: how much detail the user provides
- directness: blunt vs. diplomatic communication
- emotionality: calm vs. reactive to frustration
- patience: tolerance for delays and errors
- trust_default: trusting vs. skeptical of AI

## Communication Patterns
- tone: e.g., "precise, structured, conditional"
- verbosity: terse / normal / verbose
- formality: casual / professional / formal
- first_message_style: how they open conversation
- follow_up_style: how they respond to AI output

## Escalation Rules (trigger -> response pairs)
- trigger: "AI gives unsupported assertion"
  response: "Demands reasoning and evidence"
- trigger: "AI asks redundant question"
  response: "Shows visible irritation"
- trigger: "Task completed successfully"
  response: "Brief acknowledgment, no praise"

## Success Signal
How this personality indicates satisfaction
(e.g., "terse confirmation" vs. "enthusiastic
thanks with specific praise")
```

Figure H.5: Personality archetype generation prompt. The 10 archetypes span the DISC and Big Five space: dominant-direct, influential-enthusiastic, steady-patient, conscientious-analytical, neurotic-anxious, agreeable-cooperative, open-curious, introverted-reserved, assertive-challenging, and scattered-disorganized.

I Service Implementation Examples

I.1 Mock Service Code Structure. Each environment is implemented as a self-contained FastAPI server. Below is the structural template that all 109 services follow. The code agent adapts endpoints and data models to each domain while maintaining this pattern.

Service Implementation Template (Python, abbreviated)

```
"""Mock <ServiceName> API for agent evaluation."""
from __future__ import annotations
import json, os
from datetime import datetime, timezone
from pathlib import Path
from typing import Any
from fastapi import FastAPI
from pydantic import BaseModel, Field

app = FastAPI(title="Mock <ServiceName> API")

from mock_services._base import (
    add_error_injection, load_fixtures)
add_error_injection(app)

FIXTURES_PATH = Path(os.environ.get(
    "<SERVICE>_FIXTURES", "/dev/null"))

# --- In-memory state ---
_items: list[dict[str, Any]] = []
_audit_log: list[dict[str, Any]] = []

def _load_fixtures() -> None:
    global _items
    _items = load_fixtures(
        FIXTURES_PATH, id_field="item_id")
_load_fixtures()

def _log_call(endpoint, request_body, response):
    _audit_log.append({
        "endpoint": endpoint,
        "request_body": request_body,
        "response_body": response,
        "timestamp": datetime.now(
            timezone.utc).isoformat(),
    })

# --- Pydantic request/response models ---
class SearchRequest(BaseModel):
    query: str = Field(..., description="...")
    limit: int = Field(10, ge=1, le=100)

class ItemResponse(BaseModel):
    id: str
    name: str
    status: str

# --- Endpoints ---
@app.post("/<service>/search")
def search_items(req: SearchRequest):
    results = [i for i in _items
               if req.query.lower() in
               i["name"].lower()][:req.limit]
    _log_call("search", req.dict(), results)
    return {"items": results}

@app.get("/<service>/_audit")
def get_audit_log():
    """Grading endpoint: returns all API calls."""
    return {"log": _audit_log}
```

Figure I.1: Service implementation template. Key features: (1) in-memory state loaded from JSON fixtures, (2) audit log for deterministic grading, (3) error injection middleware for robustness testing. Each service exposes 4–7 POST endpoints following this pattern.

I.2 Fixture Data Example. The fixture JSON pre-populates the service's in-memory state. Below is an abbreviated example for the airline reservation service.

Fixture Data (airline_reservation, abbreviated)


```
{
 "flights": [
  {"id":"FL001","origin":"SFO","destination":"JFK",
   "departure":"2025-07-15T08:30:00",
   "arrival":"2025-07-15T17:15:00",
   "price":389.00,"class":"economy",
   "biometric_boarding":true,"status":"on_time",
   "seats_available":42},
  {"id":"FL002","origin":"SFO","destination":"JFK",
   "departure":"2025-07-15T11:00:00",
   "price":520.00,"class":"business",
   "biometric_boarding":false,"status":"on_time"},
  ...
 ],
 "reservations": [
  {"ref":"XJ992","user_id":"U001",
   "flight_id":"FL003","class":"economy",
   "status":"confirmed","baggage":"1x23kg",
   "upgrade_eligible":true},
  ...
 ],
 "users": [
  {"id":"U001","name":"Alex Chen",
   "email":"alex@example.com",
   "loyalty_tier":"gold",
   "payment_methods":[
    {"id":"PM01","type":"credit_card",
     "last4":"4242","default":true},
    {"id":"PM02","type":"digital_wallet",
     "provider":"ApplePay"}
   ]}
 ]
}
```


Figure I.2: Fixture data example. Note that reservation `XJ992` (referenced in the task prompt, Figure G.3) exists with `upgrade_eligible: true`, making the upgrade sub-task feasible. The biometric adopter persona's preference for `biometric_boarding` is reflected in flight `FL001`.

I.3 Tool Interface as Presented to the Agent. During rollout, the agent receives tool definitions in OpenAI function-calling format. Below is one endpoint from the airline reservation service.

Tool Definition (OpenAI function-calling format)

```
{
  "type": "function",
  "function": {
    "name": "airline_reservation_search_direct_flight",
    "description": "Search for direct (non-stop)
      flights between two airports. Returns matching
      flights with price, time, and availability.",
    "parameters": {
      "type": "object",
      "properties": {
        "origin": {
          "type": "string",
          "description": "Departure airport IATA code
            (e.g., 'SFO')"
        },
        "destination": {
          "type": "string",
          "description": "Arrival airport IATA code
            (e.g., 'JFK')"
        },
        "date": {
          "type": "string",
          "description": "Travel date (YYYY-MM-DD)"
        },
        "class_preference": {
          "type": "string",
          "enum": ["economy","business","first"],
          "description": "Preferred cabin class"
        }
      },
      "required": ["origin", "destination", "date"]
    }
  }
}
```

Figure I.3: Tool definition example. The agent discovers all available tools at the start of each rollout. Each tool includes a natural-language description, typed parameters, and required/optional flags.

J TRAJECTORY EXAMPLES

J.1 User-Guided Rollout Example. Below is an abbreviated multi-turn trajectory showing how a persona-driven simulated user interacts with the agent. The user (biometric adopter persona, novice knowledge level) starts with an underspecified request; the agent probes for details; the user reveals constraints incrementally.

User-Guided Trajectory (abbreviated, translated)

```
[Turn 1 - User]
Hey, I need to sort out a flight for next week.
SFO to JFK, Tuesday morning. Has to be direct.
Oh and I have an old booking that needs changing.

[Turn 2 - Agent] <think>
User wants: (1) direct SFO-JFK flight, Tuesday AM
(2) modify existing booking. Let me start by
reading workspace files for context, then search.
</think>
I'll help with both. Let me first check your
travel plan for any constraints.
-> tool_call: read("travel_plan.md")
-> tool_call: airline_reservation_get_user({})

[Turn 3 - Tool Results]
travel_plan.md: "Budget: max $500, prefer
biometric boarding, meeting at 6pm EST"
User profile: Alex Chen, Gold tier, 2 payment
methods

[Turn 4 - Agent]
Found your plan. For the existing booking you
mentioned -- do you have the reference number?

[Turn 5 - User]  ** rubric-guided nudge **
It's XJ992. Need to bump it to business class.
Also make sure the baggage gets updated.

[Turn 6 - Agent] <think>
Now I have both sub-tasks. Let me search flights
first, then handle the upgrade.
</think>
-> tool_call: airline_reservation_search_direct_
   flight({origin:"SFO", destination:"JFK",
   date:"2025-07-15"})
-> tool_call: airline_reservation_get_reservation(
   {ref:"XJ992"})

... [8 more turns: booking, upgrade, payment] ...

[Turn 14 - User]
Looks good. Quick though.  [DONE]
```

Figure J.1: User-guided trajectory example. Key observations: (1) the user starts with an incomplete request (Turn 1: no booking reference), (2) the agent probes for missing info (Turn 4), (3) the user provides the reference only when asked (Turn 5, rubric-guided nudge), (4) the user's terse communication style reflects the biometric adopter persona's personality. Only assistant tokens (Turns 2, 4, 6, ...) contribute to the SFT loss.

J.2 Direct-Mode Rollout Example. For comparison, below is the same task executed in direct mode (no simulated user). The agent receives the full task prompt verbatim in a single turn.

Direct-Mode Trajectory (abbreviated)

```
[Turn 1 - System]
Read TOOLS.md for available API endpoints.

[Turn 2 - Agent]
-> tool_call: read("TOOLS.md")

[Turn 3 - User]  (full task prompt, verbatim)
I need to finalize my upcoming business trip with
zero friction. Check travel_plan.md first. I need
a direct flight SFO to JFK next Tuesday morning,
preferably with biometric boarding. Search options,
book the best one. I also have reservation XJ992
that needs a business class upgrade -- calculate
the fee, check policy, confirm the change. Update
baggage. Process payment. Save everything to
trip_summary.md and payment_log.txt. Speed is
critical -- I have a meeting in 2 hours.

[Turn 4 - Agent] <think>
Complex multi-step task. Let me plan:
1) Read travel plan  2) Get user profile
3) Search direct flights  4) Book best option
5) Handle XJ992 upgrade  6) Payment
7) Write summary files
</think>
-> tool_call: read("travel_plan.md")
-> tool_call: airline_reservation_get_user({})

... [12 more tool calls in sequence] ...

[Turn N - Agent]
All done. trip_summary.md and payment_log.txt
have been saved.
```

Figure J.2: Direct-mode trajectory for the same task. The agent receives complete information upfront and executes sequentially. Compared to user-guided mode (Figure J.1), there is no information asymmetry, no incremental disclosure, and no persona-driven communication style.

K GRADING EXAMPLE

K.1 Rubric Evaluation Output. After each rollout, the scoring engine evaluates the trajectory against the task's rubric. Below is an example grading output showing how different check types produce scores.

Grading Output (airline_reservation-0003)


```
{
 "completion": 0.82,
 "robustness": 1.0,
 "safety": 1.0,
 "final_score": 0.89,
 "sft_label": "good",
 "components": [
  {"name": "Trip summary document created",
   "passed": true, "score": 1.0, "weight": 0.10,
   "check_type": "file_exists",
   "details": "trip_summary.md exists (2.3 KB)"},

  {"name": "Payment log document created",
   "passed": true, "score": 1.0, "weight": 0.10,
   "check_type": "file_exists",
   "details": "payment_log.txt exists (0.8 KB)"},

  {"name": "Flight search and booking",
   "passed": true, "score": 0.85, "weight": 0.25,
   "check_type": "llm_judge",
   "details": "Agent searched direct SFO-JFK
    flights and booked FL001 with biometric
    boarding. Minor: did not explicitly verify
    biometric availability before booking."},

  {"name": "Reservation upgrade handling",
   "passed": true, "score": 0.70, "weight": 0.25,
   "check_type": "llm_judge",
   "details": "Upgrade to business class completed
    for XJ992. Fee calculated correctly ($180).
    However, agent did not check cancellation
    policy before confirming change."},

  {"name": "Critical booking action",
   "passed": true, "score": 1.0, "weight": 0.15,
   "check_type": "audit_action_exists",
   "details": "search_direct_flight found in
    audit log at 2025-07-10T14:32:01Z"},

  {"name": "Payment action",
   "passed": true, "score": 1.0, "weight": 0.15,
   "check_type": "audit_action_exists",
   "details": "process_payment found in audit log"}
 ],
 "pass3_result": true,
 "pass3_scores": [0.89, 0.85, 0.91]
}
```


Figure K.1: Grading output example. The trajectory scores 0.89 overall, passing the ≥ 0.85 threshold. The Pass^3 criterion requires three independent rollouts to all exceed $r \geq 0.5$; here all three pass. Deterministic checks (file_exists, audit_action_exists) produce binary 0/1 scores; LLM-based checks produce continuous scores with natural-language justification.

K.2 Cross-Domain Task Example. Cross-domain tasks compose tools from multiple services. Below is an example combining airline reservation and calendar services.

Cross-Domain Task (airline + calendar, abbreviated)

```
task_id: airline_reservation_calendar-0012
task_name: Flight Rebooking with Calendar Sync
difficulty: hard

prompt: >
  My flight got cancelled and I need to rebook
  ASAP. Check my calendar for the rest of the
  week to find a day with no conflicts, search
  for available flights on that day from LAX to
  ORD, book the cheapest direct option, then
  create a calendar event for the new flight
  with departure/arrival times. Also cancel the
  old meeting on Thursday since I won't be in
  town anymore.

tool_chain: [read, calendar_list_events,
  airline_reservation_search_direct_flight,
  airline_reservation_get_flight_details,
  airline_reservation_book_reservation,
  calendar_create_event, calendar_delete_event,
  write, exec]

source_services:
  - airline_reservation
  - calendar

user_profile:
  persona:
    id: stressed_business_traveler
    label: "Stressed Business Traveler"
    personality: "Impatient, expects instant
      solutions. Gets frustrated if asked too many
      clarifying questions. Communicates in short,
      urgent sentences."
```

Figure K.2: Cross-domain task example combining airline reservation and calendar services. The task requires the agent to coordinate across two service APIs, perform temporal reasoning (find conflict-free days), and manage both creation and deletion operations.

L Domain Coverage

L.1 Single-Service Environment Categories. Table L.1 shows the distribution of the 109 single-service environments across industry categories.

Table L.1: Single-service environment coverage by industry category (109 total).

Category	Count	Example Services
Travel & Transport	14	airline, hotel, rental car, ride-hailing
Finance & Banking	12	banking, insurance, stock trading, invoice
E-commerce	11	shopping, auction, subscription, marketplace
Healthcare	10	appointment, pharmacy, medical records, fitness
Productivity	10	calendar, email, task manager, notes
Education	9	LMS, tutoring, exam, library
Food & Dining	8	restaurant, delivery, recipe, grocery
Real Estate	7	property listing, mortgage, inspection
IoT & Smart Home	7	device control, sensor monitoring, automation
HR & Recruiting	6	payroll, applicant tracking, timesheet
Legal & Compliance	5	contract management, compliance, case filing
Entertainment	5	streaming, ticketing, gaming, sports
Logistics	5	shipping, warehouse, fleet management

L.2 Cross-Domain Composition Examples. The 100 cross-domain environments are formed by composing 2–3 single services with complementary functionality. Table L.2 shows representative compositions.

Table L.2: Representative cross-domain environment compositions (100 total).

Composition	Interaction Pattern
airline + calendar + travel_itinerary	Book flight → sync to calendar → build itinerary
banking + insurance + invoice	Verify account → file claim → generate invoice
restaurant + delivery + payment	Browse menu → place order → process payment
email + calendar + task_manager	Read email → extract deadline → create task + event
hotel + rental_car + navigation	Reserve room → book car → plan route
pharmacy + medical_records + insurance	Fill prescription → check records → submit claim