Beyond Static Evaluation: Building Simulation Environments for Scalable Agentic Reinforcement Learning

Akshay Arora, Ishan Nigam, Ashutosh Aggarwal, Shefali Bansal, Krishna Singh
Sweta Kumari, Nikhil Mittal, Shariq Farhan, Siddarth Malreddy
Uber AI Solutions
San Francisco, CA, USA
smalreddy@uber.com

Abstract

As Large Language Models (LLMs) evolve into autonomous agents, traditional static evaluation fails to capture multi-step decision-making. We introduce *AgenticAI-Supervisor*, an API and UI-driven RL Gym environment that decouples environment creation from scalable execution. By moving to verifiable execution outcomes, the platform generates high-fidelity traces and applies multi-dimensional reward shaping. Critically, our framework mitigates reward hacking through rigorous internal state validation and testing. This work provides a first look at our platform's core capabilities through a Customer Support Agent case study demonstrating a consistent closed-loop feedback for model optimization. Future work will focus on advanced features such as Computer Use, Tool Use, automated "stumping", and edge-case generation.

ACM Reference Format:
Akshay Arora, Ishan Nigam, Ashutosh Aggarwal, Shefali Bansal, Krishna Singh, Sweta Kumari, Nikhil Mittal, Shariq Farhan, Siddarth Malreddy. 2026. Beyond Static Evaluation: Building Simulation Environments for Scalable Agentic Reinforcement Learning. In *Proceedings of Workshop on RL for Evaluation (RL-Eval '26).* ACM, New York, NY, USA, 5 pages.

1 Introduction

Large Language Models (LLMs) are transitioning from conversational interfaces into autonomous agents capable of reasoning across external tools and complex GUI-based applications [1, 8, 17]. Unlike traditional chatbots, these agents operate in dynamic environments where success depends on long-horizon planning and error recovery [6]. However, as agentic workflows expand, static single-turn benchmarks fail to capture the multi-step decision-making and environmental feedback these agents require [1]. Consequently, enterprise-grade models exhibit a severe reliability gap, failing approximately 76% of complex professional tasks due to compounding execution errors [8, 14]. In high-stakes operations such as supply chain auditing or procurement, enterprises cannot rely on systems prone to losing logical consistency or violating implicit constraints over long horizons [9].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
RL-Eval '26, San Jose, CA, USA
© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.

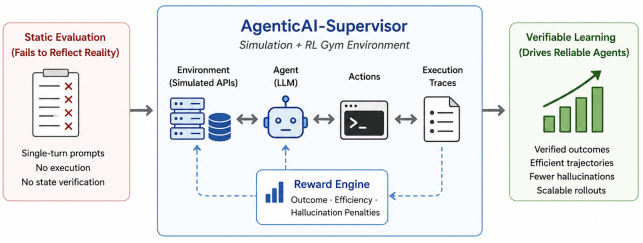

Figure 1: AgenticAI Supervisor enables scalable, verifiable reinforcement learning by combining simulated environments, execution traces, and multi-dimensional rewards.

To establish trust in autonomous systems, the evaluation paradigm must shift from grading textual responses to verifying programmatic actions within large-scale reinforcement learning (RL) lifecycles [16]. While execution traces enable deep-dives into reasoning failure modes [5], the manual authorship of multi-step test cases, "stumping" prompts, and edge scenarios remains an error-prone bottleneck [4]. This creates a scaling deficit, necessitating foundational infrastructure that can automate high-fidelity environment generation and bridge the gap between subjective heuristics and verifiable state validation.

To address these challenges, we introduce the *AgenticAI-Supervisor*, a foundational API and UI-driven simulation environment engineered for continuous agent evaluation and optimization [7]. Operating as a high-fidelity ground-truth engine, the platform enables thousands of parallel, isolated trajectories within a secure sandbox where agents interact with simulated tools and web-based UIs [7, 15]. By integrating simulation directly into the authoring pipeline, we establish a scalable "Run-to-Verify" loop that ensures 100% data integrity and generates the verifiable rewards necessary for deep RL [15]. Our framework specifically mitigates reward hacking through rigorous internal state-mutation testing, ensuring agent trajectories align with business logic rather than exploiting heuristic gaps. Our core contributions are:

- A dual-phase framework that decouples synthetic environment scaffolding from high-concurrency rollout execution.
- A deterministic reward shaping engine that moves beyond heuristic evaluation to strict state validation, penalizing hallucinations and enforcing trajectory efficiency.
- A case study of an autonomous Customer Support agent, demonstrating our efficacy in generating high-fidelity traces for multi-step constraint reasoning and planning.

2 Related Work

This section reviews recent literature on evaluating and optimizing autonomous agents. We focus on the shift toward interactive simulation environments (Section 2.1), the adoption of Reinforcement Learning (RL) for multi-step workflows (Section 2.2), and the use of automated reward shaping via deterministic and LLM-based verifiers (Section 4.3).

2.1 From Static Benchmarks to Interactive Environments

Historically, LLMs have been evaluated using static, single-turn benchmarks (e.g., MMLU or GSM8K), which measure isolated textual predictions or reasoning capabilities without environmental feedback. However, as LLMs are increasingly deployed as autonomous agents, these static metrics fail to capture the multi-step decision-making, planning, and tool-use required for real-world tasks [6, 17]. To bridge this gap, interactive evaluation frameworks and benchmarks such as ALFWorld, WebShop, and WebArena were introduced. These platforms simulate partially observable environments where agents must navigate state changes and adapt to dynamic feedback over time [13, 14]. The *AgenticAI-Supervisor* extends these paradigms by focusing on high-fidelity, enterprise-specific API emulations rather than generalized web browsing or game-based tasks, providing a verifiable "Run-to-Verify" sandbox.

2.2 Reinforcement Learning for Agentic Workflows

While Supervised Fine-Tuning (SFT) is effective for teaching models basic tool-calling syntax and chat templates, it is insufficient for training robust agents capable of error recovery and long-horizon planning [3, 12]. Reinforcement Learning (RL) has emerged as the standard for optimizing agentic behavior across entire trajectories. Rather than forcing a model to mimic a static "golden path," modern RL techniques such as Proximal Policy Optimization (PPO) and Group Relative Policy Optimization (GRPO) allow the agent to explore reasoning paths and receive feedback based on verifiable outcomes [12]. A critical challenge in multi-turn RL is environment stabilization and state management; our platform addresses this by executing isolated rollouts via stateless container jobs that interact with deterministic mock databases.

2.3 Automated Evaluation and Reward Shaping

A primary bottleneck in scaling RL for agentic systems is the design of reliable reward functions. In closed-loop systems, deterministic verifiers which check for database mutations, verify final costs against budget constraints, or pass unit tests provide the most robust reward signals resistant to reward gaming [3]. However, for open-ended or qualitative interactions, recent frameworks increasingly rely on "LLM-as-a-Judge" mechanisms. These models evaluate trajectories at the turn-level or episode-level to provide scalar rewards or detect major deviations [2, 11]. Our architecture incorporates both: strict programmatic verifiers for environment state validation and trajectory efficiency, combined with LLM judges to penalize hallucinations and enforce soft constraints.

3 Simulation and Execution Framework

Generating high-fidelity reinforcement learning (RL) data requires environments that transition beyond static synthesis to functional real-world representation. To prevent performance divergence between simulation and production, our architecture decouples environment instantiation from large-scale execution. This systemic separation ensures that agent evaluations remain representative of actual operational constraints.

3.1 High-Fidelity Environment Scaffolding

Enterprise workflows involve complex dependencies and non-deterministic error paths that are difficult to replicate in static simulations. To capture this functional fidelity, our framework utilizes three core components:

- **Agentic Workflows:** Domain-driven execution paths that incorporate standard operations alongside deliberate failure states, missing data, and ambiguous tool responses to test agent resilience.
- **Base Tool Simulator:** Reusable infrastructure for stateful tools spanning backend APIs and interactive web-based UIs exposed via the Model Context Protocol (MCP) [10] specification for tool interface standardization.
- **Dataset Connectors:** A state management layer that binds test cases to specific environmental contexts ensuring consistent initialization and grounding for each rollout.

3.2 Scalable Execution Engine and Rollout Orchestration

The Test Run Engine operationalizes parallel rollouts within isolated, stateless sandboxes to ensure statistical significance. The Rollout Handler provisions containerized instances with a discrete lifecycle, preventing state leakage between iterations. Within the environments, the Agent Runtime orchestrates the interaction loop incorporating LLM prompting, action parsing for tool calls and GUI interactions, and observation retrieval. For observability, discrete events (invocations, executions, reward assignments) are logged as structured *Spans*. Aggregations of these spans form high-fidelity execution *Traces* used for debugging and reward modeling.

4 Closed-Loop Reward Formulation for Reinforcement Learning

The reward framework combines terminal verification criteria with a continuous trajectory efficiency signal, providing both sparse task-completion feedback and dense behavioral quality signal for policy optimization.

4.1 Execution Trace Repository

All rollout data including reward scores, per-criterion subscores, and intermediate spans is committed upon episode completion. Each span encodes a discrete event (LLM invocation, tool call, state change, reward assignment). Their aggregation forms a complete execution trace for offline debugging and batch sampling.

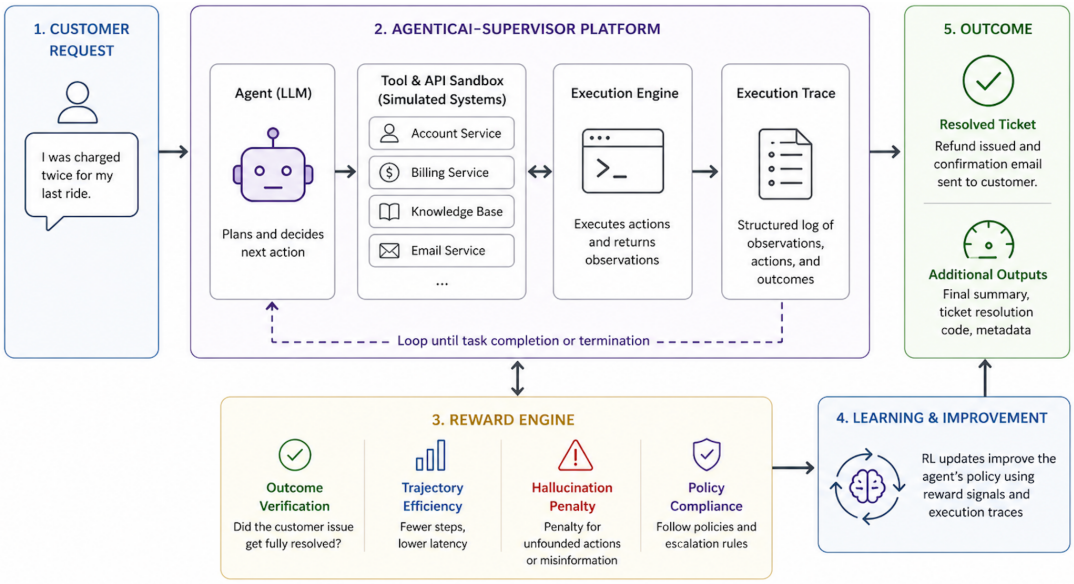

Figure 2: Autonomous customer support built using AgenticAI-Supervisor. A customer request is handled by an LLM agent operating inside a simulated support environment with read-only tools for account, billing, policy, and communication lookup, and actionable tools for resolution workflows. Each interaction is executed inside the sandbox, logged as an execution trace, and scored by a reward engine based on outcome verification, trajectory efficiency, hallucination penalties, and policy compliance. These signals drive iterative reinforcement learning to improve resolution quality and reliability over time.

4.2 Multi-Dimensional Reward Shaping

The reward framework targets three dimensions, each addressing a distinct failure mode observed in deployed agentic systems. Terminal criteria such as **Outcome Reward** and **Constraint Adherence** supply sparse binary signals on whether the agent reached a valid final state. These are complemented by a continuous **Trajectory Efficiency Reward** component that provides dense process-based feedback spanning tool correctness, call redundancy, API error rate, required-tool coverage, and step economy.

Outcome Reward The terminal environment state is compared against curated *golden answers* via multiset equality over normalized action keys. Additionally, the committed state must satisfy a configurable pre-determined resource budget. This primary task-completion signal is binary and is blind to the agent's natural language output and trajectory quality.

Constraint Adherence Agents may satisfy outcome criterion through strategies that violate implicit behavioral constraints. Real-world analysis (detailed in Sec. 5) found constraint misrepresentation in ~40% of positively reinforced episodes and fabrication in ~3.8% of episodes under outcome only rewards, motivating explicit constraint verification. Constraint Adherence enforces **(i)** negative checks: records must not assume forbidden field values, **(ii)** side-effect detection: entity counts at episode end are compared against setup-time baselines catching reward hacking via spurious record creation, and **(iii)** output fidelity: fabricated factual claims in the agent's response are cross-referenced against tool API responses in the trace.

Trajectory Efficiency Reward This is the primary process-based signal in our reward formulation and evaluates behavioral quality by directly using the structured tool calls, GUI events, and result records. It consists of the following five sub-components:

- *Tool Correctness*: Individual tool invocations are scored for validity independently of overall trajectory shape. The reward linearly combines the fraction of valid (non-error, non-banned) calls with a penalty for invoking explicitly prohibited actions.
- *Redundant Call Penalty*: A tool invocation is redundant if the same tool is called with identical parameters after a prior successful invocation already exists in the trace, inflating trajectory length without adding information.
- *Validation Error Penalty*: The proportion of tool invocations returning an error response, penalizing malformed parameters and invalid API usage.
- *Min-Tool Coverage Score*: Ensures that the required tools are invoked according to task specifications, penalizing both under-calling and excessive usage. A smooth non-linear function blends the deficit and excess penalties, which is then sigmoid-aggregated with redundancy and error penalties to produce a continuous unit-normalized trajectory efficiency score.
- *Step-Penalized Efficiency Modifier*: Penalizes unnecessarily long rollouts by scaling the reward by the difference between the executed and ground truth traces. A configurable unit-bound hyperparameter controls the rate of decay providing a differentiable verbosity penalty. This incentivizes agents to reach correct terminal states via minimal reasoning chains.

4.3 Deterministic and LLM-as-Judge Verifiers

Evaluation is implemented through two modalities. **(1) Verifiable Rewards** perform deterministic, state-based checks such as matching golden answers, validating environment constraints, and cross-referencing identifiers across API and GUI traces without incurring inference costs. Qualitative dimensions, including response coherence and reasoning quality, are assessed via **(2) LLM-as-a-Judge** evaluators using structured rubrics. This modality provides a dense partial-credit signal based on the fraction of essential criteria met, rather than a binary pass/ fail evaluation. To ensure consistency, we employ ensemble judging to reduce evaluator variance with the final reward balance being calibrated via configuration to suit specific environment requirements.

5 Case Study: Autonomous Customer Support

To validate the simulation environment's efficacy, an Agentic AI Reinforcement Learning Environment was designed around a Customer Support domain, tasking the agent with autonomously resolving customer issues, processing transactions, and managing account security.

5.1 Emulated Domain Tools and API Integration

The emulated environment features a bifurcated suite of API and GUI tools, categorized into "Actionable" and "Non-Actionable" (Read-Only) tools. This structure necessitates that the agent gathers context before executing state-mutating actions.

The non-actionable toolkit provides observational capabilities. The agent utilizes `get_customer_info` and `get_order_details` to retrieve comprehensive customer profiles and order histories. To ensure consistent support and policy compliance, the agent can verify past communications via `check_interaction_history` and cross-reference company guidelines using `search_kb_and_policies`.

Once sufficient context is gathered, the agent invokes actionable tools to mutate the environment state:

- **Financial & Fulfilment Tools:** The **Refund Tool** processes monetary returns, while the **Replacement Tool** resolves defective item reports by creating a new fulfillment order; both automatically update order statuses and log actions to the associated ticket.
- **Security Tool:** Locks accounts for identified fraud (e.g., `account_takeover` or `return_fraud`), cancels all pending orders, and creates an audit trail.
- **Workflow Management:** Tools such as **create_ticket** and **update_order_status** allow the agent to manage the lifecycle of customer grievances and order logistics.

5.2 Evaluation Suite and Scenario Complexity

Evaluation was conducted against a curated suite of tasks designed to test the agent's ability to sequence these tools effectively. Complex scenarios required the agent to cross-reference knowledge base policies before authorizing refunds, or to detect suspicious activity in the interaction history to trigger appropriate security locks. By generating continuous, verifiable reward signals across these simulations, the platform demonstrates its capacity to reinforce safe, policy-compliant customer support workflows.

6 Discussion and Future Work

We outline a roadmap to transition from foundational infrastructure to a comprehensive ecosystem for autonomous agent optimization.

6.1 No-Code Simulation Interfaces

Future work will externalize "Gym Factory" capabilities via a unified portal for self-serve environment customization and out-of-the-box deployment. This no-code dashboard will allow domain experts to utilize drag-and-drop scenario builders to configure mocked tools, bind datasets, and define reward strategies. This will significantly reduce the lead time for deploying specialized RL gyms.

6.2 Human-in-the-Loop Reward Overrides

While deterministic verifiers and LLM-as-a-judge mechanisms provide scalable reward signals for well-defined tasks, human oversight remains essential for nuanced, high stakes, or consequential decisions. To address this, we plan to integrate Human-in-the-Loop (HITL) workflows, allowing experts to override automated rewards with qualitative feedback directly in the training pipeline. Additionally, an expert marketplace will validate synthesized environments for realism and solvability prior to large-scale deployment.

6.3 Automated "Stumping"

To refine frontier models, agents must navigate increasingly difficult scenarios. Future iterations will automate "stumping" the systematic generation of hard task variants by injecting failure modes, ambiguous states, and missing data. This automation accelerates the creation of robust training datasets, preparing agents for the stochasticity of live production.

6.4 Uncertainty-Aware Reward Signals

A promising direction involves incorporating *semantic entropy* into the reward pipeline to quantify response unreliability without supervision. By measuring divergence across multiple rollouts, high-entropy states can serve as penalty signals or curriculum markers to prioritize complex tasks. This leverages our trace-based infrastructure to identify inconsistent action distributions. Overall, the goal is a platform that autonomously scales environments with minimal effort, establishing a foundation for reliable enterprise agents.

7 Conclusion

As models transition into enterprise agents, static evaluation is no longer sufficient; what matters is whether an agent can act successfully inside a realistic environment. By standardizing trace-based evaluation and verifiable rewards through the *AgenticAI-Supervisor*, we provide the essential training ground where agents can act, fail, and improve in a secure sandbox before ever touching production data. Our framework specifically mitigates reward hacking by ensuring that terminal rewards are tied to internal state validation rather than surface-level textual heuristics. While this paper provides a first look at our foundational architecture, future work will focus on sharing more advanced features - such as automated "stumping" - and integrating human-in-the-loop (HITL) workflows with self-serve, no-code UI capabilities to democratize the rapid creation of custom RL gyms.

References

[1] Anonymous. 2026. ReliabilityBench: Evaluating LLM Agent Reliability Under Production-Like Stress Conditions. *arXiv preprint* (2026). https://arxiv.org/pdf/2601.06112 Quantifies how pass@1 overestimates agent reliability by 20-40%..

[2] L. Chen et al. 2025. Human or Agent-Automated Judging: Evaluating LLMs. *arXiv preprint* (2025).

[3] Fireworks AI. 2025. Best Practices for Multi-Turn RL. https://fireworks.ai/blog/best-practices-for-multi-turn-RL

[4] J. Gao et al. 2025. Scaling Verifiable Rewards for Agentic Workflows via Automated Scenario Synthesis. *arXiv preprint arXiv:2510.12000* (2025). Discusses the automation of "stumping" prompts to overcome the manual test-creation bottleneck..

[5] S. Lee et al. 2026. From Traces to Policies: High-Fidelity Feedback Loops for Autonomous Tool-Use. *Journal of Machine Learning Research (JMLR)* (2026). Examines the use of structured execution traces to optimize multi-step agent reasoning..

[6] A. Mohammadi et al. 2025. Conceptual Foundations and Taxonomy of LLM Agent Evaluation Frameworks. *arXiv preprint* (2025).

[7] T. Muller et al. 2026. The Run-to-Verify Paradigm: Ensuring Data Integrity in Synthetic RL Environments. *preprint arXiv:2602.11000* (2026). Defines the "Run-to-Verify" framework for closed-loop reward signal generation..

[8] Molisha Shah. 2026. Multi-Agent AI Systems: Why They Fail and How to Fix Coordination Issues (MAST Taxonomy). *NeurIPS 2025 Datasets and Benchmarks Track* (2026). Analyzes 1,600+ traces showing 41-86% failure rates in production..

[9] Sierra Research. 2026. τ^2-Bench: Realistic Benchmarking for Dual-Control Agents in Enterprise Domains. *Scientific Report* (2026). Simulates complex customer service and retail operations with dual-control environment mutations..

[10] David Soria Parra and Justin Spahr-Summers. 2024. Model Context Protocol (MCP). https://modelcontextprotocol.io.

[11] Weiting Tan, Xinghua Qu, Ming Tu, Meng Ge, Andy T. Liu, Philipp Koehn, and Lu Lu. 2026. Process-Supervised Reinforcement Learning for Interactive Multimodal Tool-Use Agents. In *ICLR*.

[12] Unsloth AI. 2026. Reinforcement Learning environments and how to build them. https://unsloth.ai/blog/rl-environments

[13] Z. Wang et al. 2026. A Survey on Evaluation of LLM-based Agents. *arXiv preprint arXiv:2503.16416* (2026).

[14] F. Xu et al. 2025. TheAgentCompany: Benchmarking LLM Agents on Consequential Real World Tasks. *arXiv preprint arXiv:2412.14161* (2025).

[15] Y. Zhang et al. 2026. Massively Parallel Simulation for Agentic Reinforcement Learning. *Journal of Artificial Intelligence Research* (2026). Discusses the infrastructure required for high-concurrency rollout execution in LLM agents..

[16] R. Zheng et al. 2026. Scaling Laws for Agentic Reinforcement Learning in Enterprise Environments. *ICML 2026* (2026). Provides empirical evidence for the transition from prompt engineering to industrial RL lifecycles..

[17] Y. Zhu et al. 2025. Distinctions Between Static Chatbots and Agents: A Survey. *arXiv preprint* (2025).