The Environment Layer: Building Infrastructure for Agentic AI Training

A PREPRINT

Fei Wang **Eric Wang** **Salon Ren** **Michael Zhang**
Alchedata AI

April 10, 2026

Keywords Agentic AI, Reinforcement Learning Environments, RL Gyms, Environment Design, Synthetic Environments, Sim-to-Real Transfer, POMDP, Environment Infrastructure

Abstract

The emergence of Agentic Reinforcement Learning (Agentic RL) has created an urgent need for sophisticated training environments that go beyond traditional RL benchmarks. While conventional LLM-RL operates within single-step MDPs, Agentic RL requires environments supporting multi-turn interactions, tool integration, and verifiable reward signals. This white paper argues that **RL environments are the foundational infrastructure for agentic AI**—the critical layer that determines what capabilities agents can learn and how reliably they transfer to deployment.

We present a comprehensive analysis of the environment layer, organized around three core questions: (1) **Environment Design**—what makes an effective Agentic RL environment, including observation spaces, action interfaces, and reward mechanisms; (2) **Environment Infrastructure**—the frameworks, protocols, and tools for building and deploying environments at scale; and (3) **Environment Quality**—methodologies for evaluating environment fidelity, the sim-to-real gap, and production readiness.

We survey the ecosystem of environment frameworks (OpenEnv, GEM, MCP), synthetic environment generation pipelines (Agent World Model, Reasoning Gym), and specialized environments for embodied AI (NVIDIA Cosmos, Isaac Sim). We introduce the Environment Quality Framework (EQF) for systematic environment evaluation and analyze the critical sim-to-real gap through the User-Sim Index. Finally, we present a research agenda for next-generation RL environments that will enable the transition from research prototypes to production-ready agentic systems.

Contents

1 Introduction

The rapid convergence of Large Language Models (LLMs) and Reinforcement Learning (RL) has precipitated a fundamental transformation in how AI systems are conceived, moving from "answering" to "acting." However, this transformation exposes a critical infrastructure gap: while model capabilities have advanced dramatically, the **environments** in which these models learn to act remain underdeveloped. This white paper argues that RL environments—not just algorithms or model architectures—are the foundational bottleneck and opportunity in Agentic AI.

1.1 The Environment-Centric Perspective

Traditional surveys of Agentic RL focus on model capabilities, algorithms, and applications. We take a different approach: we center the **environment layer** as the primary determinant of what agents can learn and how reliably they transfer to deployment. This perspective is motivated by a simple observation: the most sophisticated policy architecture cannot compensate for an environment that fails to capture the essential dynamics of real-world tasks.

Consider the analogy to software development. Compilers, debuggers, and testing frameworks form the infrastructure layer that enables reliable software engineering. Similarly, RL environments form the infrastructure layer for training autonomous agents. Just as a compiler that fails to catch memory errors leads to unreliable software, an environment that fails to model edge cases leads to unreliable agents.

This stakes are high: in enterprise-style workflows, long sequences with the wrong design can accumulate small per-step errors, creating almost a domino effect. Very quickly, a 1% error rate per step can compound to a 63% chance of failure by the hundredth step [Nuñez, 2025]. This observation underscores that environment design does not only have a strong impact on RL outcomes—it can make an entire training process useless if the environment is poorly specified.

1.2 Why Environments Matter More Than Ever

The shift from single-turn LLM interactions to multi-turn agentic behavior fundamentally changes environment requirements:

From Static to Dynamic. Conventional LLM-RL operates within a degenerate, single-step MDP where the environment state is static (the prompt). Agentic RL requires environments with dynamic state evolution, where agent actions have persistent consequences across extended interaction horizons.

From Subjective to Objective Rewards. Traditional RLHF optimizes for subjective human preferences. Agentic RL requires environments with *verifiable* reward signals—objective, automatable criteria that determine task success without human judgment.

From Isolated to Integrated. Agents must interact with diverse external systems: code interpreters, web browsers, databases, and APIs. Environments must provide standardized interfaces for tool integration while maintaining safety and reproducibility.

1.3 The Environment Quality Problem

A critical challenge that has received insufficient attention is **environment quality**. Not all training environments are created equal. We identify three dimensions of environment quality:

1. **Fidelity**: How accurately does the environment model real-world dynamics?
2. **Verifiability**: Can success be objectively and automatically determined?
3. **Diversity**: Does the environment provide sufficient variation for generalization?

Research using the User-Sim Index (USI) reveals that LLM-based user simulators often create an "easy mode" for agents, exhibiting behavioral gaps (excessive cooperativeness, information front-loading) and evaluative gaps (uniformly positive feedback). This sim-to-real gap represents a fundamental challenge: agents trained on low-fidelity environments may appear competent during training but fail catastrophically in deployment.

Treating environments as a "contract" between algorithms and world dynamics is one practical approach to clarifying environment design responsibilities [Abaka AI, 2025]. This contract metaphor reinforces that the environment API defines the task precisely—not a proxy for the task, but the task itself. Gym-style APIs successfully encode this contract: `reset` initializes a task, `step` applies an action, and the environment returns the next observation plus a reward and termination signal.

1.4 Contributions and Paper Structure

This white paper provides a comprehensive analysis of the environment layer for Agentic RL. Our contributions are organized around three core questions:

1. **Environment Design** (Sections 2-3): What makes an effective Agentic RL environment? We derive design principles from POMDP theory and analyze environment requirements across capability dimensions.
2. **Environment Infrastructure** (Sections 4-7): What frameworks, protocols, and tools exist for building and deploying environments? We survey the ecosystem of environment frameworks, synthetic generation pipelines, and specialized environments for embodied AI.
3. **Environment Quality** (Sections 8-10): How do we evaluate and improve environment fidelity? We analyze the sim-to-real gap, algorithmic considerations for environment-aware training, and production readiness frameworks.

Section 11 concludes with a research agenda for next-generation RL environments. We argue that the path to production-ready agentic systems runs through the environment layer—and that investment in environment infrastructure will yield compounding returns across the entire Agentic AI stack.

1.5 Relationship to Prior Work

The closest work to ours is Wolgast [2025], a January 2025 practitioner guide on RL environment design from the perspective of *traditional* RL applied to physical systems (energy grids, robotics, locomotion). That work provides valuable low-level guidance on Gymnasium API usage, observation normalization, action-space shaping, and reward engineering—concerns that remain largely orthogonal to the challenges addressed here. Critically, it operates entirely within the classic MDP paradigm with numeric state vectors, atomic discrete or continuous actions, and single-step reward signals, and makes no reference to LLMs, language-conditioned agents, or the agentic AI stack.

Our paper departs from Wolgast in three fundamental ways. First, *scope*: we target environments specifically designed for training *language-model-based agents* executing long-horizon, multi-step tasks in digital and embodied settings—a regime where the environment must handle natural language observations, tool calls, and delayed, verifiable rewards, none of which appear in Gymnasium-style benchmarks. Second, *theory*: we formalize agentic training as a POMDP over history-conditioned policies and introduce the Expanded Action Space (ExpA) framework that decouples language generation, routing decisions, and environment interactions—a distinction irrelevant for low-dimensional control tasks. Third, *novelty*: we contribute original frameworks—EQF (Environment Quality: Fidelity, Diversity, Verifiability), CLASSic (Cost, Latency, Accuracy, Security, Stability), CLEAR (Cost-normalized accuracy), and the User-Sim Index (USI)—that quantify environment quality and production readiness for agentic systems. None of these constructs have analogues in prior environment-design literature, which has focused primarily on classical RL domains without language or agentic structure.

2 Theoretical Framework: From LLM-RL to Agentic RL

The transition to Agentic Reinforcement Learning represents a fundamental reconceptualization of how large language models interact with their environment. This section establishes the mathematical formalism underlying this paradigm shift, contrasting the degenerate single-step MDP framework of conventional LLM-RL with the temporally extended POMDP formulation of Agentic RL. We introduce the Expanded Action Space (ExpA) concept as a theoretical mechanism for decoupling reasoning from action execution.

2.1 MDP vs. POMDP Formalism

2.1.1 Conventional LLM-RL: The Degenerate MDP

Traditional reinforcement learning for large language models, typically instantiated as Preference-Based Reinforcement Fine-Tuning (PBRFT), operates within a constrained mathematical framework that can be characterized as a degenerate, single-step Markov Decision Process. In this formulation, the horizon length $H = 1$, reducing the sequential decision-making problem to an isolated prediction task.

Definition 2.1 (Conventional LLM-RL MDP). The standard LLM-RL framework is defined as a tuple $\mathcal{M}_{\text{LLM}} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ where:

- **State space** $\mathcal{S}$: Consists of static text prompts $s \in \mathcal{S}$, where each prompt represents a fixed context with no temporal evolution.

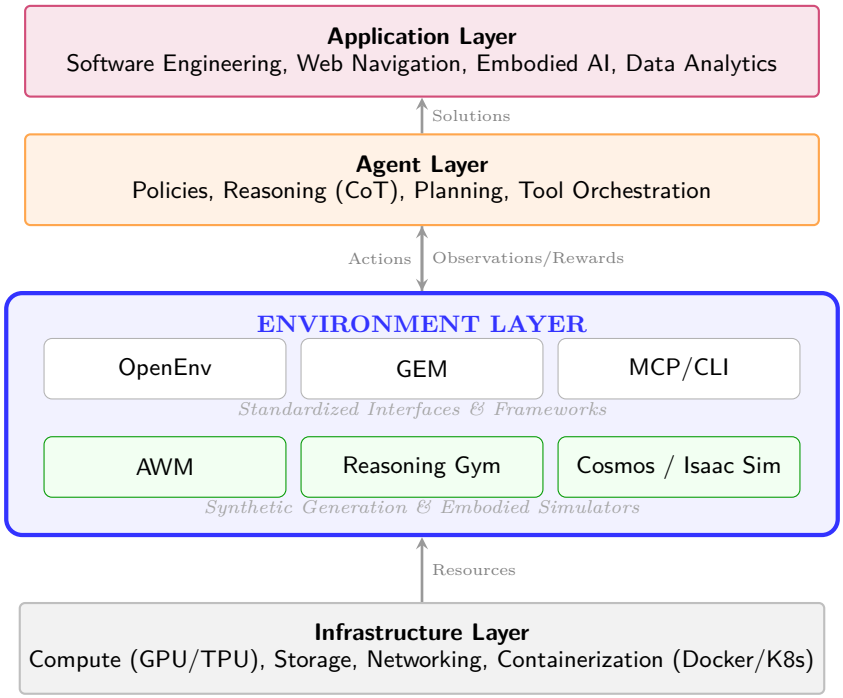


Figure 1: The Environment Layer as the foundational infrastructure for Agentic AI. Positioned between low-level compute infrastructure and high-level agent capabilities, the environment layer provides standardized interfaces (OpenEnv, GEM, MCP), synthetic generation pipelines (AWM, Reasoning Gym, Cosmos/Isaac), and quality assurance mechanisms. This layer determines what capabilities agents can learn and how reliably they transfer to deployment.

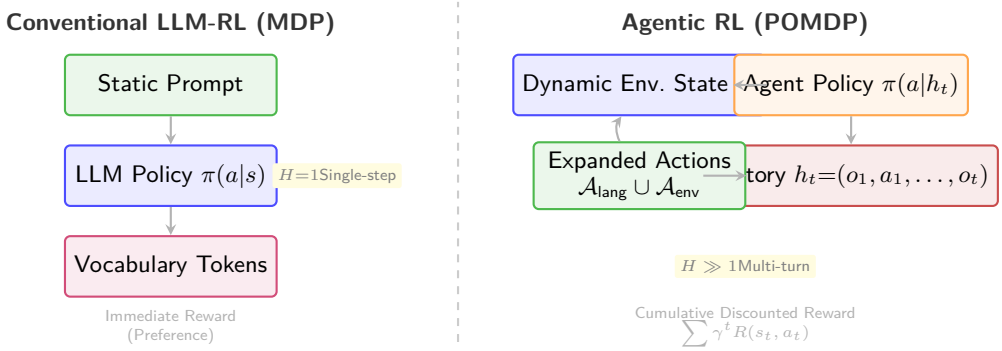


Figure 2: The fundamental paradigm shift from conventional LLM-RL to Agentic RL. (Left) Conventional LLM-RL operates as a single-step MDP with static prompts, vocabulary-only actions, and immediate rewards. (Right) Agentic RL formalizes interaction as a multi-turn POMDP with dynamic states, expanded action spaces including tool use, and cumulative rewards over extended horizons.

- **Action space** $\mathcal{A} = \mathcal{V}$: The vocabulary tokens, where $a_t \in \mathcal{V}$ represents the next token prediction.
- **Transition function** $\mathcal{T}(s'|s, a) = \delta(s' - s)$: The environment state remains static; the prompt does not change based on the model's output.
- **Reward function** $\mathcal{R}(s, a)$: Typically derived from human preference models or reward models trained on preference data, providing a scalar signal $r \in \mathbb{R}$.
- **Discount factor** γ: Usually irrelevant for single-step decisions ($\gamma \in [0, 1]$).

The objective in this framework reduces to maximizing immediate reward:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{s\sim\mathcal{D},a\sim\pi(\cdot|s)}[\mathcal{R}(s, a)]$$

This formulation treats the LLM as an open-loop generator, where each response is produced without consideration of future interactions or the consequences of its outputs on subsequent states.

2.1.2 Agentic RL: The POMDP Formulation

Agentic RL reframes the interaction between language models and their environment as a temporally extended Partially Observable Markov Decision Process (POMDP), capturing the essential characteristics of autonomous agency: persistent state, partial information, and long-horizon planning.

Definition 2.2 (Agentic RL POMDP). The Agentic RL framework is defined as a POMDP $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O}, \gamma, H)$ where:

- **State space** $\mathcal{S}$: The environment state $s_t \in \mathcal{S}$ evolves dynamically based on agent actions and external dynamics. Unlike the static prompts of LLM-RL, these states represent the complete configuration of the world.
- **Action space** $\mathcal{A}$: Expanded beyond vocabulary tokens to include tool invocations, API calls, and environment manipulations (detailed in Section 2.2).
- **Transition function** $\mathcal{T}(s_{t+1}|s_t, a_t)$: Governs how the environment state evolves in response to agent actions, capturing the causal structure of the world.
- **Reward function** $\mathcal{R}(s_t, a_t, s_{t+1})$: Provides task-specific feedback, often sparse and delayed, indicating progress toward objectives.
- **Observation space** Ω: The agent does not directly observe the full state s_t but receives observations $o_t \in \Omega$ through the observation function.
- **Observation function** $\mathcal{O}(o_t|s_t, a_{t-1})$: Defines the probability of observing o_t given the current state and previous action, formalizing partial observability.
- **Discount factor** $\gamma \in [0, 1)$: Determines the present value of future rewards.
- **Horizon** $H \gg 1$: The episode length extends over multiple interaction steps.

2.1.3 State Space: Environment State and Agent Memory

A critical distinction between LLM-RL and Agentic RL lies in the composition of the agent's effective state. In the POMDP formulation, the agent maintains an internal representation that aggregates information across the interaction history.

Definition 2.3 (Agent Information State). At time t, the agent's information state h_t encompasses the complete history of prior interactions:

$$h_t = (o_0, a_0, o_1, a_1, \ldots, o_{t-1}, a_{t-1}, o_t) \in \mathcal{H}_t$$

where $\mathcal{H}_t = (\Omega \times \mathcal{A})^t \times \Omega$ represents the space of all possible histories of length t.

The agent's policy $\pi(a_t|h_t)$ conditions on this history rather than the current observation alone. In practice, LLM-based agents implement this through:

- **Context window**: The concatenation of prior observations and actions within the model's context window serves as an implicit memory mechanism.
- **External memory**: Structured storage systems (e.g., RAG, vector databases) that the agent can query to retrieve relevant historical information.

- **Learned state representations**: Internal representations maintained by the model that compress historical information into a latent state.

The objective in Agentic RL is to maximize the cumulative discounted reward over the full horizon:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{H-1} \gamma^t \mathcal{R}(s_t, a_t, s_{t+1})\right]$$

where the expectation is taken over the initial state distribution, transition dynamics, observation emissions, and policy-induced action distributions.

2.1.4 Partial Observability and the Observation Function

The observation function $\mathcal{O}(o_t|s_t, a_{t-1})$ formalizes the limited information available to the agent. In Agentic RL contexts, partial observability arises from several sources:

1. **Information asymmetry**: The environment may contain state variables invisible to the agent (e.g., hidden files in a filesystem, internal database states).
2. **Sensing limitations**: The agent's perception is constrained to what can be captured through available interfaces (e.g., screenshots, API responses, text outputs).
3. **Delayed effects**: Actions may have consequences that are not immediately observable, requiring the agent to infer latent state changes.

The belief state $b_t(s) = P(s_t = s|h_t)$ provides a probabilistic representation of the agent's uncertainty about the true environment state given its history. Optimal behavior in POMDPs often requires maintaining and updating this belief state, though LLM-based agents typically employ heuristic approximations through their context windows and reasoning capabilities.

Observation design choices are themselves a theory of what matters in a task [Abaka AI, 2025]. This theory should be made explicit and normalized for stable learning, with noise or partiality added only when it improves transfer to the real deployment distribution. A practical example comes from autonomous-driving simulation: LiDAR sensors have explicit configurable parameters (e.g., number of channels, points per second), as well as explicit models for point drop-off and distance noise. Whether to match real sensor noise, exaggerate noise for robustness, or remove it for learnability is a deliberate design choice that changes what the agent learns.

Table 1 summarizes the core shift from preference-tuned, single-step language modeling to long-horizon agent-environment interaction.

Property	**Conventional LLM-RL (PBRFT)**	**Agentic Reinforcement Learning**
Mathematical Setup	Single-step MDP ($H = 1$)	Multi-turn POMDP ($H \gg 1$)
State Representation	Static text prompt	Dynamic environment plus memory
Action Space	Vocabulary tokens ($\mathcal{A} = \mathcal{V}$)	Tokens plus tool, API, and environment actions
Reward Signal	Subjective preference (alignment)	Objective task completion (utility)
Information	Full observability	Partial observability
Temporal Structure	Stateless, independent decisions	Stateful, sequential dependencies

Table 1: Comparison between conventional LLM-RL and Agentic Reinforcement Learning under the POMDP formulation.

2.2 Expanded Action Spaces (ExpA)

2.2.1 Decoupling Reasoning from Action

A theoretical advancement in Agentic RL is the formal separation of internal reasoning processes from external environment interactions through the concept of Expanded Action Spaces (ExpA). This decoupling addresses a fundamental limitation of treating language generation and environment manipulation as identical operations.

Definition 2.4 (Expanded Action Space). The expanded action space $\mathcal{A}_{\text{ExpA}}$ is defined as the union of three disjoint subspaces:

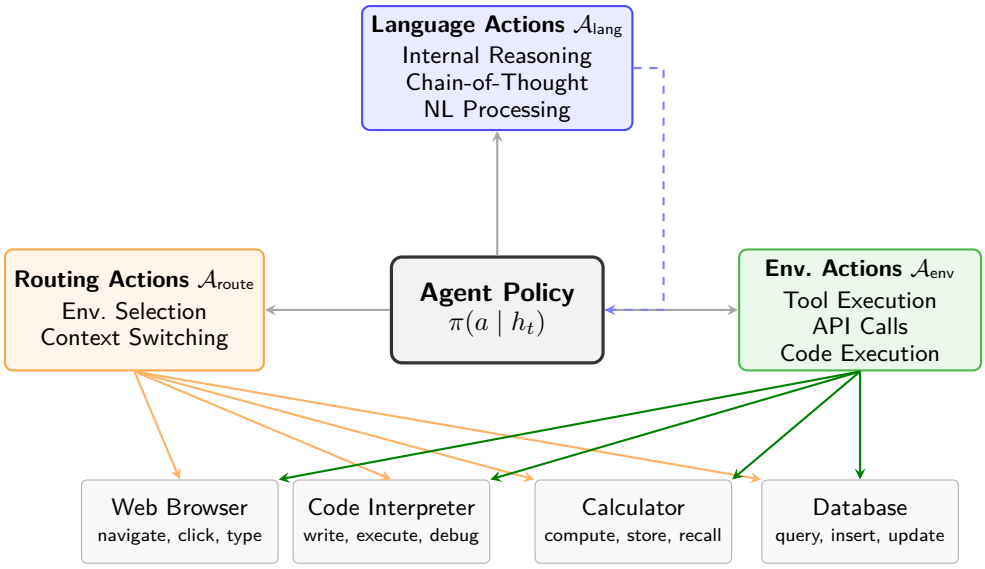


Figure 3: The Expanded Action Space (ExpA) framework decouples reasoning from execution through three action types. Language actions ($\mathcal{A}_{\text{lang}}$) support internal reasoning; routing actions ($\mathcal{A}_{\text{route}}$) select environments; environment actions ($\mathcal{A}_{\text{env}}$) execute tools and APIs.

$$\mathcal{A}_{\text{ExpA}} = \mathcal{A}_{\text{lang}} \cup \mathcal{A}_{\text{route}} \cup \mathcal{A}_{\text{env}}$$

where:

- **Language actions** $\mathcal{A}_{\text{lang}} = \mathcal{V}^*$: Sequences of vocabulary tokens used for internal reasoning, chain-of-thought generation, and natural language processing. These actions affect only the agent's internal state (context window) and produce no external effects.
- **Routing actions** $\mathcal{A}_{\text{route}}$: Discrete selector actions that determine which environment or tool the agent will interact with next. A routing action $a_{\text{route}} \in \mathcal{A}_{\text{route}}$ activates a specific environment interface.
- **Environment actions** $\mathcal{A}_{\text{env}} = \bigcup_{e \in \mathcal{E}} \mathcal{A}_e$: The union of action spaces for all available environments $\mathcal{E}$. When environment e is active, actions $a \in \mathcal{A}_e$ are interpreted according to that environment's semantics.

2.2.2 Routing Actions and Environment Selection

Routing actions enable the agent to dynamically switch between different operational modes and external systems. Formally, we define a routing function:

$$\rho : \mathcal{A}_{\text{route}} \to \mathcal{E}$$

that maps routing actions to environment instances. When the agent executes a routing action a_{route} at time t, the subsequent action a_{t+1} is interpreted within the context of environment $\rho(a_{\text{route}})$.

Example 2.1 (Multi-Environment Routing). Consider an agent with access to three environments:

- $\mathcal{E}_{\text{web}}$: Web browser environment with actions $\mathcal{A}_{\text{web}} = \{\text{navigate}, \text{click}, \text{type}, \text{scroll}, \ldots\}$
- $\mathcal{E}_{\text{code}}$: Code execution environment with actions $\mathcal{A}_{\text{code}} = \{\text{write\_file}, \text{execute}, \text{debug}, \ldots\}$
- $\mathcal{E}_{\text{calc}}$: Calculator environment with actions $\mathcal{A}_{\text{calc}} = \{\text{compute}, \text{store}, \text{recall}, \ldots\}$

The routing action space $\mathcal{A}_{\text{route}} = \{a_{\text{web}}, a_{\text{code}}, a_{\text{calc}}\}$ allows the agent to switch contexts:

1. Agent generates reasoning: $a_t \in \mathcal{A}_{\text{lang}}$ ("I need to calculate the sum first...")
2. Agent routes to calculator: $a_{t+1} = a_{\text{calc}} \in \mathcal{A}_{\text{route}}$
3. Agent executes calculation: $a_{t+2} = \text{compute}(x + y) \in \mathcal{A}_{\text{calc}}$

Aspect	Traditional LLM-RL	ExpA-Based Agentic RL
Token Function	All tokens are outputs	Tokens partitioned by function
Environment Interface	Implicit through text	Explicit routing mechanism
Action Semantics	Uniform (all are text)	Context-dependent (language versus environment)
Extensibility	Requires retraining for new capabilities	Modular addition of environments
Observability	Black-box output	Transparent reasoning and action separation

Table 2: Comparison between traditional LLM-RL and ExpA-based Agentic RL.

4. Agent routes back to reasoning: $a_{t+3} = a_{\text{lang}} \in \mathcal{A}_{\text{route}}$

2.2.3 Theoretical Implications of ExpA

The Expanded Action Space framework provides several theoretical advantages:

Modularity: By separating $\mathcal{A}_{\text{lang}}$, $\mathcal{A}_{\text{route}}$, and $\mathcal{A}_{\text{env}}$, the agent's reasoning capabilities can be trained and optimized independently of specific environment implementations. This modularity enables:

$$\pi(a|h) = \pi_{\text{route}}(a_{\text{route}}|h) \cdot \pi_{\text{env}}(a_{\text{env}}|h, a_{\text{route}}) \cdot \pi_{\text{lang}}(a_{\text{lang}}|h)$$

where the policy decomposes into routing, environment-specific, and language components.

Composability: New environments can be added to $\mathcal{E}$ without retraining the core reasoning capabilities, provided the routing mechanism can map to the new action space.

Interpretability: The explicit separation of reasoning tokens ($\mathcal{A}_{\text{lang}}$) from action tokens ($\mathcal{A}_{\text{env}}$) enables process-level supervision and debugging. The agent's "thought process" becomes observable through its language actions, while its external effects are traceable through environment actions.

Safety: Routing actions provide natural intervention points for guardrails and safety checks. Before executing $a_{\text{env}} \in \mathcal{A}_e$, the system can validate the action against environment-specific constraints.

These design principles for action spaces connect directly to the practical requirements identified in agentic training environments [Abaka AI, 2025]: (1) **Validity** ensures the agent cannot perform impossible or meaningless actions that accidentally yield reward; (2) **Granularity** recognizes that higher-level action representations often outperform direct low-level commands; and (3) **Safety constraints** enforce boundaries around humans, hardware, and resource limits. The key difference between a "working" action space and a "trainable" one is not whether it is mathematically valid, but whether it makes correct behavior easy to express and incorrect behavior hard to stumble into.

2.2.4 Comparison with Traditional LLM-RL

The ExpA framework fundamentally differs from traditional LLM-RL in how it treats the relationship between language and action:

As shown in Table 2, the crucial difference is that ExpA turns tool use and environment selection into first-class policy decisions rather than requiring text to implicitly encode actions.

In traditional LLM-RL, when a model generates the token sequence "CALCULATE 2+2", the system must parse this text to extract the intended action. In the ExpA framework, the model explicitly routes to $\mathcal{E}_{\text{calc}}$ and executes compute$(2 + 2)$, eliminating the ambiguity of text-based action specification.

The mathematical formulation of Agentic RL with Expanded Action Spaces provides a rigorous foundation for understanding how large language models can evolve from passive generators into autonomous agents capable of sustained interaction with complex, dynamic environments. This framework enables the development of training methodologies, evaluation protocols, and safety mechanisms appropriate for the agentic paradigm.

3 Taxonomy of Agentic Capabilities and Task Domains

Progress in Agentic Reinforcement Learning is best understood through a systematic classification of the core capabilities that enable autonomous behavior and the task domains where these capabilities are exercised. This taxonomy provides a structured framework for analyzing agentic systems, identifying gaps in current research, and guiding future development efforts.

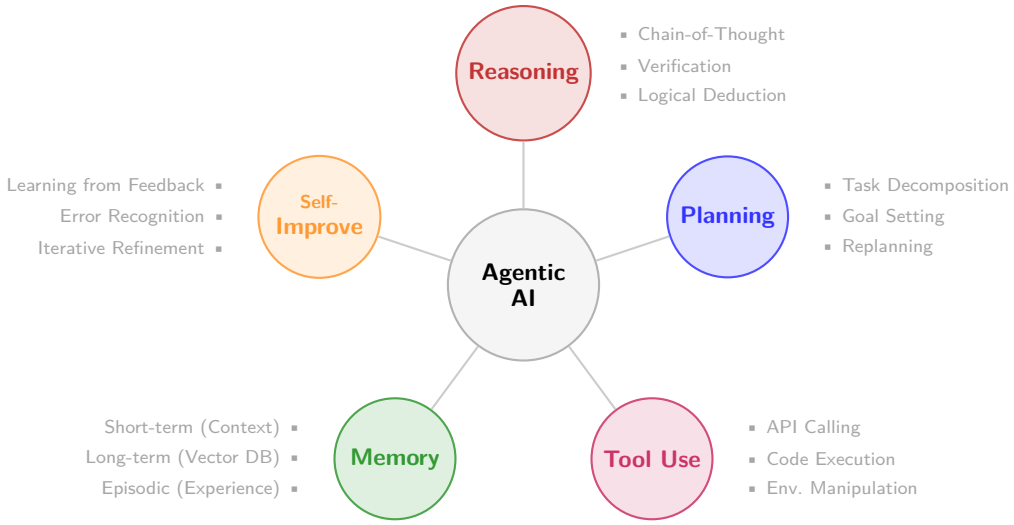


Figure 4: A taxonomy of the five core agentic capabilities required for autonomous operation: Planning (decomposition, goal setting, replanning), Tool Use (API calling, code execution), Memory (short-term, long-term, episodic), Self-Improvement (learning from feedback, error recognition), and Reasoning (chain-of-thought, verification).

3.1 Core Agentic Capabilities

The transition from passive language models to autonomous agents requires the development of five interconnected capabilities: planning, tool use, memory, self-improvement, and reasoning. Each capability represents a distinct dimension of agency that must be cultivated through appropriate training environments and reinforcement learning algorithms.

3.1.1 Planning

Planning constitutes the foundational capability that enables agents to decompose complex objectives into manageable sub-goals and coordinate actions across extended time horizons. In the Agentic RL framework, planning operates through several mechanisms:

Task Decomposition refers to the agent's ability to break down complex, multi-step objectives into discrete, actionable sub-tasks. For example, when asked to "build a web application," an agent must decompose this into requirements analysis, architecture design, implementation, testing, and deployment phases. RL environments that support planning must provide intermediate reward signals that validate correct decomposition while penalizing redundant or misordered steps.

Goal Setting involves the agent's capacity to establish intermediate milestones that guide behavior toward ultimate objectives. Effective goal setting requires the agent to maintain a hierarchical representation of objectives, balancing immediate rewards against long-term utility. Training environments must support variable goal structures, allowing agents to learn adaptive goal-setting strategies across different problem domains.

Replanning represents the agent's ability to adjust its strategy when encountering unexpected obstacles or changed circumstances. This requires monitoring execution progress, detecting deviations from expected outcomes, and generating alternative action sequences. Replanning capability is particularly critical in dynamic environments where initial plans may become obsolete due to external changes.

3.1.2 Tool Use

Tool use extends the agent's action space beyond vocabulary tokens to include interactions with external systems, APIs, and computational resources. This capability transforms language models from isolated text generators into systems capable of affecting real-world change.

API Calling involves the agent's ability to invoke external services through structured interfaces. This includes understanding API documentation, constructing valid requests, parsing responses, and handling errors. Training for API calling requires environments that simulate realistic API behaviors, including rate limiting, authentication failures, and malformed responses.

Code Execution enables agents to write, execute, and debug programs to accomplish computational tasks. This capability is essential for data analysis, algorithmic problem-solving, and system administration. Effective training environments must provide sandboxed execution contexts that capture execution outputs, runtime errors, and resource consumption metrics.

The Expanded Action space (ExpA) framework formalizes tool use by decoupling environment interactions from language generation. Under this approach, the model maintains a clean separation between reasoning tokens and action tokens, with "routing actions" triggering transitions between different interaction modes.

3.1.3 Memory

Memory systems enable agents to persist information across interaction episodes and retrieve relevant context for current decision-making. Without effective memory, agents are limited to purely reactive behaviors, unable to learn from past experiences or maintain consistency across extended sessions.

Short-term Memory encompasses the immediate context window that the agent can directly attend to during inference. While modern models support increasingly long contexts, effective short-term memory utilization requires training to identify and retain task-relevant information while filtering distractions.

Long-term Memory refers to structured storage systems that persist beyond individual sessions, typically implemented through vector databases or knowledge graphs. Agents must learn to encode experiences into retrievable formats and construct appropriate queries to access stored information.

Episodic Memory captures specific interaction sequences that can be referenced for future similar situations. This form of memory is particularly valuable for learning from rare events or capturing successful solution patterns that can be reused across tasks.

3.1.4 Self-Improvement

Self-improvement represents the meta-capability that enables agents to enhance their own performance through experience. This capability distinguishes truly autonomous systems from those that remain static after deployment.

Learning from Feedback involves updating behavior based on explicit reward signals, implicit user satisfaction indicators, or self-generated evaluations. Effective self-improvement requires environments that provide dense, informative feedback signals that correlate with true task performance.

Error Recognition enables agents to identify when their outputs are incorrect or suboptimal, even in the absence of external feedback. This capability is crucial for initiating self-correction cycles and avoiding the propagation of errors through multi-step reasoning chains.

Iterative Refinement refers to the process of progressively improving solutions through multiple attempts, incorporating lessons from previous failures. Training environments that support self-improvement must allow for multi-attempt episodes with rich feedback on the quality of intermediate solutions.

3.1.5 Reasoning

Reasoning capabilities enable agents to perform explicit, verifiable inference steps that lead to correct conclusions. Unlike implicit pattern matching, reasoning produces interpretable chains of thought that can be validated and debugged.

Chain-of-Thought reasoning involves generating explicit intermediate steps between problem statement and solution. This approach, often implemented through `<think>` tags or similar mechanisms, makes the reasoning process visible and subject to process-level supervision.

Verification capabilities enable agents to check the correctness of their own reasoning steps or final outputs. This includes mathematical verification, logical consistency checking, and cross-referencing against known facts. Training for verification requires environments that provide ground-truth labels or automated checking procedures.

3.2 Task Domains

The capabilities described above find application across diverse task domains, each presenting unique challenges and requiring specialized environment designs. Three domains have emerged as particularly important proving grounds for Agentic RL research.

3.2.1 Software Engineering

Software engineering tasks require agents to understand codebases, implement features, fix bugs, and maintain systems over time. This domain is characterized by:

- **Structured Action Spaces**: Code modifications follow syntactic and semantic constraints
- **Objective Verification**: Test suites provide unambiguous success criteria
- **Long Horizons**: Complex tasks may require hundreds of sequential actions
- **Tool Integration**: Effective development requires using version control, build systems, and testing frameworks

Benchmarks like SWE-Bench provide standardized evaluation environments where agents must resolve real GitHub issues in production codebases. Success in this domain requires strong planning capabilities for architecture design, tool use for build and test execution, and reasoning capabilities for understanding complex code relationships.

3.2.2 Mathematical Reasoning

Mathematical reasoning tasks evaluate an agent's ability to solve problems requiring precise, step-by-step logical deduction. Key characteristics include:

- **Verifiable Solutions**: Mathematical claims can be checked for correctness with certainty
- **Symbolic Manipulation**: Solutions often require algebraic or logical transformations
- **Multi-step Inference**: Complex problems require chains of deductions
- **Abstraction**: Solutions may require recognizing patterns or applying general theorems

Benchmarks like AIME (American Invitational Mathematics Examination) and various olympiad problems provide challenging test cases. The Reasoning Gym framework offers procedurally generated mathematical tasks that enable curriculum learning and infinite training data generation.

3.2.3 Web and GUI Interaction

Web and GUI interaction tasks require agents to navigate visual interfaces, understand layout semantics, and perform actions through simulated user inputs. This domain features:

- **Multimodal Perception**: Agents must process visual information alongside text
- **Dynamic Environments**: Web pages and applications change state in response to actions
- **Implicit Constraints**: Interface design conventions must be inferred from context
- **Real-world Relevance**: Success transfers directly to practical automation tasks

Benchmarks like OSWorld and BrowserGym provide simulated environments where agents interact with operating systems and web browsers. Success requires integrating visual perception with planning and tool use capabilities.

This benchmark family is broader than the headline suites usually cited in agent papers. MiniWoB++ remains useful for evaluating atomic browser skills such as clicking, typing, and menu navigation; WebArena and BrowserGym focus on realistic multi-page workflows; and ALFWorld, TextWorld, and ScienceWorld extend the environment lens to grounded text-based action spaces where partial observability, exploration, and long-horizon planning can be studied under controlled conditions.

3.3 Capability-Domain Classification Matrix

The following matrix summarizes how core capabilities map to requirements across task domains:

Table 3 makes clear that environment design is inseparable from evaluation design: each domain rewards a different combination of planning, tool use, memory, and reasoning.

3.4 Toward a Diagnostic Benchmark: Testing Capabilities in Isolation

The capability-domain matrix surfaces a fundamental measurement problem: existing benchmarks evaluate *end-to-end task success*, which conflates all five capabilities. A failure on SWE-Bench could reflect inadequate planning, insufficient tool-use fidelity, memory overflow, inability to self-correct, or flawed reasoning—yet aggregate task completion rates cannot distinguish these failure modes. This measurement gap limits both reproducibility of results and the ability to design targeted training interventions.

We propose a **Capability Diagnostic Examination (CaDEx)**, a structured suite of micro-benchmarks that tests each capability in isolation by holding all others constant:

Capability	Software Engineering	Mathematical Reasoning	Web/GUI Interaction
Planning	Architecture design and task decomposition	Problem decomposition and proof strategy	Navigation planning and task sequencing
Tool Use	Compilers, debuggers, and version control	Calculators, theorem provers, and symbolic systems	Browser APIs, form interactions, and click actions
Memory	Codebase understanding and pattern libraries	Formula retrieval and prior problem solutions	Session state, user preferences, and history
Self-Improvement	Bug-fix patterns and refactoring strategies	Proof-technique learning and error-pattern recognition	Interface adaptation and failure recovery
Reasoning	Logic verification and type checking	Deductive chains and symbolic manipulation	Visual reasoning and layout inference

Table 3: Capability-domain matrix showing how core agentic capabilities manifest across major benchmark families.

1. **L0 — Single-step Tool Fidelity.** Tasks requiring exactly one tool call with a verifiable, deterministic result (e.g., a SQL SELECT, a file read, an API call with known response). All planning, memory, and reasoning demands are removed. Tests Tool Use in isolation; a well-functioning agent should score >90% at L0. Persistent failure here indicates fundamental action-space misspecification, not a reasoning deficit.
2. **L1 — Sequential Planning Under Complete Observability.** Tasks requiring a k-step plan in a fully observable, deterministic environment where the complete world state is provided on every observation. Isolates Planning by eliminating partial observability and memory pressure. Difficulty is parameterized by plan depth k; accuracy vs. k curves reveal whether the agent can maintain goal coherence over extended decomposition chains.
3. **L2 — Long-Horizon Retrieval Under Context Pressure.** Conversation tasks where the agent must correctly reference information provided exactly N turns earlier, with intervening turns containing distractors. Tests Memory by controlling retrieval distance N; the primary metric is Recall@N as a function of N. Degradation slopes reveal whether memory failures are attributable to context length limits or retrieval strategy.
4. **L3 — Adversarial Error Injection.** An otherwise-solvable task is interrupted at step t by a planted error (malformed tool response, corrupted file, contradictory observation). Tests Self-Improvement by measuring whether the agent detects the fault, diagnoses its source, and repairs the execution without restarting from scratch. Recovery rate vs. injection depth t/T is the primary metric.
5. **L4 — Proof-and-Verify Reasoning Chains.** Tasks requiring the agent to (a) generate a step-by-step derivation and (b) verify its own output against an automated ground-truth checker. Tests Reasoning by separating generation from verification; the verify-match rate—fraction of generations that pass the checker after self-review—isolates the agent's ability to catch its own reasoning errors.
6. **L5 — Full Integration.** Standard long-horizon benchmark tasks (SWE-Bench, WebArena, OS-World). Performance here, when combined with L0–L4 profiles, enables *failure attribution*: the gap between L5 and the minimum of L0–L4 scores is an estimate of the *integration overhead* arising from capability interference.

Capability Curriculum Protocol (CCP). CaDEx profiles motivate a principled **curriculum learning** strategy. Agents trained directly on long-horizon tasks often develop brittle, shortcut-exploiting policies because sparse terminal rewards provide no signal for distinguishing capability bottlenecks. The CCP prescribes a staged training schedule gated by CaDEx-measured proficiency:

Stage 1. **Foundation.** Train on L0 tasks across all required tool categories. Gate: >90% L0 success across tools before advancing.

Stage 2. **Sequential.** Train on L1 tasks at $k = 3$, then $k = 5$, then $k = 10$. Gate: >80% step accuracy at each depth.

Stage 3. **Persistence.** Introduce L2 memory tasks with $N = 10$, scaling to $N = 50$. Gate: Recall@50 >70%.

Level	Capability Isolated	Key Control Variable	Primary Metric
L0	Tool Use	Tool type and API complexity	Single-call success rate
L1	Planning	Plan depth k	Step accuracy vs. k
L2	Memory	Retrieval distance N	Recall@N
L3	Self-Improvement	Error injection depth t/T	Recovery rate vs. t/T
L4	Reasoning	Derivation length and proof complexity	Verify-match rate
L5	Integration	Full task (SWE-Bench, WebArena, etc.)	Task success rate

Table 4: The CaDEx (Capability Diagnostic Examination) protocol. Each level tests one capability in isolation by holding all others constant. L5 integration scores combined with L0–L4 profiles enable failure attribution beyond aggregate task success.

Stage 4. **Recovery.** Introduce L3 error injection at $t/T = 0.25$, then 0.5, then 0.75. Gate: Recovery rate >60% at each injection depth.

Stage 5. **Verification.** Train on L4 proof-and-verify tasks. Gate: Verify-match rate >75%.

Stage 6. **Integration.** Graduate to L5 end-to-end tasks. Monitor the integration overhead; if L5 performance falls more than 20 points below the minimum of L0–L4 scores, diagnose which capability combination is failing and re-train the relevant stage.

This structured approach connects environment design back to the EQF framework: environments used in each CCP stage can be scored on the EQF rubric to ensure that fidelity, diversity, and verifiability are appropriate for the capability being trained. L0 environments require high verifiability but need not be highly diverse; L5 environments should meet the full production threshold of EQF ≥ 7.0.

3.5 Concrete Examples

To illustrate the taxonomy in practice, consider how an agentic system might approach a representative task from each domain. These examples are also aligned with benchmark families commonly used to evaluate agentic systems, such as SWE-Bench for software engineering, AIME-style mathematical reasoning tasks, and WebArena or OSWorld-style interactive workflows.

Software Engineering Example (SWE-Bench-style): Given a real GitHub issue in a production repository where a dependency upgrade has broken the CI test suite, the agent must:

1. *Plan*: Break the issue into reproduction, root-cause analysis, patch design, regression testing, and validation against the original bug report
2. *Use Tools*: Search the codebase, inspect git history, run the failing tests, edit source files, and execute the full test suite
3. *Memory*: Retain knowledge of repository structure, prior failed hypotheses, and project-specific conventions discovered during exploration
4. *Self-Improve*: Refine the debugging strategy after unsuccessful patches, using test feedback and stack traces to narrow the search space
5. *Reason*: Infer how the dependency change altered program behavior, identify the minimal corrective patch, and verify that the fix does not introduce regressions

Mathematical Reasoning Example (AIME-style): Solving an AIME-style number theory problem with a single exact answer requires:

1. *Plan*: Identify the relevant invariants, choose a proof strategy, and decompose the problem into intermediate lemmas or subcases
2. *Use Tools*: Invoke symbolic or numerical computation to test conjectures, verify modular arithmetic, and check boundary cases
3. *Memory*: Recall similar olympiad techniques, intermediate deductions, and previously eliminated cases without repeating work
4. *Self-Improve*: Detect when a line of reasoning leads to contradiction, revise assumptions, and switch to a more promising strategy

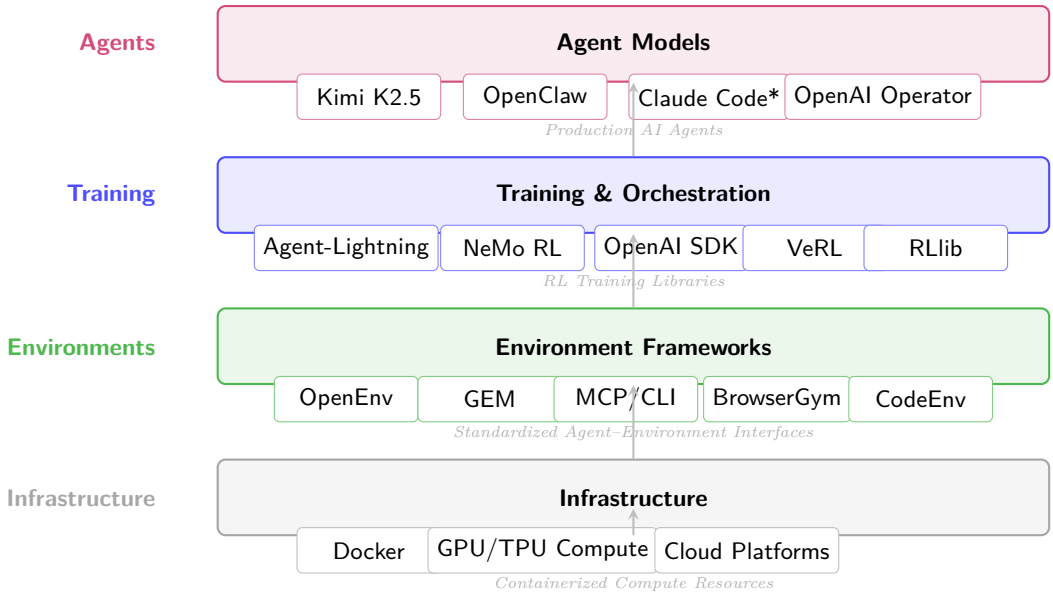


Figure 5: The four-layer architecture of the Agentic RL ecosystem. Infrastructure Layer provides containerized compute; Environment Layer (OpenEnv, GEM, MCP) standardizes agent-environment interfaces; Training Layer (Agent-Lightning, NeMo RL, OpenAI SDK) orchestrates learning; Agent Layer highlights representative agent systems (Kimi K2.5, OpenClaw, Claude Code*), where * denotes a proprietary product example included for comparison.

5. *Reason*: Construct a rigorous derivation that leads to the unique final answer and confirm consistency between the proof and computational checks

Web Interaction Example (WebArena/OSWorld-style): Resolving a customer support request in a browser-based commerce dashboard requires:

1. *Plan*: Sequence the workflow across order lookup, policy verification, refund or replacement processing, and confirmation logging
2. *Use Tools*: Navigate the web UI, search records, fill forms, click controls, and extract relevant information from dynamic pages
3. *Memory*: Track the current session state, the customer's order details, and the actions already taken across multiple pages
4. *Self-Improve*: Recover from failed searches, unexpected pop-ups, or changed layouts by trying alternate navigation paths
5. *Reason*: Interpret interface cues, apply business rules correctly, and verify that the final action satisfies the support request without violating policy

This taxonomy provides a framework for analyzing existing systems, identifying capability gaps, and designing targeted training interventions. As the field progresses, we expect the boundaries between capabilities to blur as agents develop increasingly integrated and fluid competence across all dimensions.

4 Packages, Protocols, and Development Frameworks

The rapid maturation of Agentic Reinforcement Learning has been catalyzed by a robust ecosystem of protocols, publicly available SDKs, research codebases, and vendor-led libraries. Some of these systems are open source, while others are open standards or proprietary products with public APIs and documentation. Together, they standardize agent-environment interactions, abstract away infrastructure complexity, and enable researchers and practitioners to focus on algorithmic innovation rather than low-level system integration. This section provides a comprehensive survey of the dominant environment frameworks and RL training libraries that constitute the foundation of modern Agentic RL development.

Feature	Description	Benefit
Containerized Isolation	Docker-based environment sandboxing	Safe execution of arbitrary code and system commands
Gymnasium Compatibility	Standardized `step()`/`reset()` API	Reuse of existing RL training infrastructure
Multi-Domain Support	Browser, code, database, and API environments	Unified training across diverse task types
Async Vectorization	Parallel environment execution	High-throughput data collection for training
Trajectory Recording	Built-in episode logging and replay	Debugging, evaluation, and demonstration collection

Table 5: Key OpenEnv design features and the training benefits they provide.

4.1 Environment and Interface Frameworks

Standardized environment frameworks serve as the critical abstraction layer between agent policies and the diverse task domains they must navigate. These frameworks provide Gymnasium-style APIs that ensure consistency across different environments while supporting the complex, multi-turn interactions characteristic of Agentic RL.

4.1.1 OpenEnv (Meta/HuggingFace)

OpenEnv represents a collaborative effort between Meta AI and Hugging Face to establish a unified, end-to-end framework for Agentic RL environments. Drawing inspiration from the widely-adopted Gymnasium (formerly OpenAI Gym) interface, OpenEnv extends the paradigm to support the unique requirements of language model agents operating in extended-horizon POMDPs.

Core Architecture and Design Principles

OpenEnv's architecture centers on containerized environment isolation, ensuring that agent actions cannot affect the host system while maintaining realistic interaction dynamics. The framework exposes a familiar API surface through three primary methods:

- `reset()`: Initializes the environment state and returns the initial observation
- `step(action)`: Executes an action and returns the tuple `(observation, reward, terminated, truncated, info)`
- `render()`: Provides human-readable visualization of the current state

This design choice enables seamless migration of existing RL algorithms while accommodating the expanded action spaces (ExpA) required for agentic capabilities. OpenEnv supports diverse domain implementations including BrowserGym for web navigation, CodeEnv for software engineering tasks, and custom environment definitions through a plugin architecture.

Key Features and Capabilities

Table 5 summarizes the infrastructure properties that make OpenEnv representative of the new environment layer.

The framework's emphasis on reproducibility and safety addresses a critical gap in Agentic RL research. By containerizing environments, OpenEnv ensures that experimental results are deterministic and that agents cannot inadvertently cause system damage during explorationa significant concern when training models capable of executing arbitrary code or making web requests.

Code Example: Basic OpenEnv Usage

```
from openenv import BrowserGymEnv
import gymnasium as gym

# Initialize the browser environment
env = BrowserGymEnv(task_name="web_shopping", headless=True)

# Reset and get initial observation
obs, info = env.reset()
```

```
# Run an episode
done = False
total_reward = 0

while not done:
    # Agent policy selects action (e.g., click, type, scroll)
    action = agent.predict(obs)

    # Execute action in environment
    obs, reward, terminated, truncated, info = env.step(action)

    total_reward += reward
    done = terminated or truncated

    # Access environment state for debugging
    if info.get("error"):
        print(f"Environment error: {info['error']}")

env.close()
print(f"Episode complete. Total reward: {total_reward}")
```

4.1.2 GEM (General Experience Maker)

The General Experience Maker (GEM) is a simulator framework specifically architected for the age of Large Language Models. Unlike traditional RL environments that were designed for low-dimensional state spaces and atomic actions, GEM embraces the complexity of language-based agents operating in asynchronous, multi-step scenarios.

Architectural Distinctions

GEM's most significant departure from conventional frameworks is its native support for asynchronous, vectorized execution. Recognizing that LLM inference is the primary bottleneck in Agentic RL training, GEM decouples environment state management from agent inference, allowing multiple environments to progress concurrently while the agent model processes observations in batch.

The framework implements a task-agnostic core that can be specialized through configuration rather than code modification. This design philosophy enables rapid prototyping of new task domains without requiring deep familiarity with the underlying simulation engine. GEM's state representation uses structured JSON schemas that can be directly consumed by LLM agents, eliminating the need for complex observation preprocessing pipelines.

Performance Characteristics

GEM's asynchronous architecture yields substantial throughput improvements over synchronous alternatives. In benchmark evaluations, GEM demonstrates:

- **5-10x higher throughput** compared to synchronous environment execution
- **Sub-10ms latency** for environment state transitions
- **Scalability to 1000+ concurrent environments** on standard compute clusters

These performance characteristics make GEM particularly well-suited for large-scale RL training runs where sample efficiency is paramount. At the same time, recent comparisons of agent-RL infrastructure position GEM closer to a tightly coupled, in-process framework than to a fully decoupled rollout service: environments are instantiated as Python objects and stepped within the training stack, which improves convenience but can complicate migration across trainers and execution backends.

4.1.3 Model Context Protocol (MCP) - Anthropic

The Model Context Protocol (MCP), introduced by Anthropic in late 2024, represents a paradigm shift in how agents connect to external tools and data sources. Rather than treating tool integration as an application-level concern, MCP establishes an open standard that functions as the "USB-C port for AI applications"a universal interface enabling seamless connectivity between agents and diverse external systems.

Protocol Specification and Design

MCP operates on a client-server architecture where:

- **MCP Clients** (agents) initiate connections and request capabilities
- **MCP Servers** expose specific functionalities (tools, resources, prompts) through a standardized protocol

The protocol defines three primitive operations:

1. **Tools**: Executable functions that agents can invoke (e.g., database queries, API calls, file operations)
2. **Resources**: Structured data sources that provide context (e.g., documentation, configuration files, logs)
3. **Prompts**: Pre-defined templates for common interaction patterns

Security and Enterprise Integration

A distinguishing feature of MCP is its emphasis on security and governance. The protocol includes built-in authentication mechanisms, permission scoping, and audit logging capabilities essential for enterprise deployment. MCP servers can enforce fine-grained access controls, ensuring that agents only access authorized resources and that all actions are traceable for compliance purposes.

4.1.4 CLI as Tool Interface: The Emerging Alternative

Despite MCP's enterprise features, a competing trend has gained significant momentum: treating CLI tools as the primary agent-tool interface rather than building dedicated MCP servers. The argument is straightforward — CLI tools already exist for most systems, require zero schema injection, and are already deeply represented in model training data.

The Token Efficiency Argument. A typical MCP server schema for a complex enterprise system (GitHub, Microsoft Graph) can consume 20,000–55,000 tokens of context before any actual task is attempted. An equivalent CLI invocation — leveraging a tool the model already knows — often costs fewer than 200 tokens including the full command and output. In one documented case, an equivalent task ran 35x cheaper in token terms using CLI versus MCP for Microsoft Graph integration.

Composability and Unix Philosophy. CLI tools compose naturally through pipes, redirection, and scripting. Agents can chain CLI tools the same way developers do, constructing pipelines from git, grep, curl, jq, and domain-specific CLIs without custom protocol layers or schema negotiation.

When MCP Wins. MCP retains advantages in scenarios demanding formal security boundaries, centralized audit logs, fine-grained permission scoping, or standardized multi-agent tool sharing. Where governance and compliance are non-negotiable, MCP's structured approach provides guarantees that ad-hoc CLI invocation cannot. MCP also enables dynamic tool discovery at runtime — an agent can enumerate and use unfamiliar tools without pre-programming.

Emerging Convergence. Rather than a winner-take-all outcome, the practical picture suggests ecosystem stratification: CLI-first design for developer-centric agent tooling where efficiency and flexibility dominate; MCP for enterprise-grade integrations where security, compliance, and cross-platform tool sharing are primary. Platforms like Microsoft's mcp-cli demonstrate this hybrid thinking, using CLI as the transport while retaining MCP-style dynamic tool discovery. The open question is whether future agent platforms will treat CLI as an implementation detail beneath a unified protocol layer, or whether CLI-native interfaces will prove sufficient for most production agent deployments.

Code Example: MCP Client Implementation

```
from mcp import ClientSession, StdioServerParameters
from mcp.client.stdio import stdio_client

# Configure connection to MCP server
server_params = StdioServerParameters(
    command="python",
    args=["mcp_server.py"],
    env={"API_KEY": "sk-..."}
)

async def use_mcp_tools():
    async with stdio_client(server_params) as (read, write):
        async with ClientSession(read, write) as session:
            # Initialize connection
```

Framework	Primary Focus	Containerization	Async Support	Protocol Standard	Best For
OpenEnv	Multi-domain RL environments	Yes (Docker)	Yes	Gymnasium API	Research and safety-critical training
GEM	High-throughput simulation	Optional	Native	Custom JSON	Large-scale training runs
MCP	Tool integration standard	N/A	Yes	MCP Protocol	Enterprise tool connectivity
CLI	Direct command invocation	N/A	Limited	Shell/POSIX	Developer-centric agent tooling

Table 6: Comparison of representative environment and interface frameworks for Agentic RL.

```
    await session.initialize()

    # Discover available tools
    tools = await session.list_tools()
    print(f"Available tools: {[tool.name for tool in tools.tools]}")

    # Invoke a tool
    result = await session.call_tool(
        name="query_database",
        arguments={"sql": "SELECT * FROM users LIMIT 10"}
    )

    return result
```

Comparison of Environment Frameworks

Table 6 compares the major framework patterns discussed in this section.

4.2 RL Training and Orchestration Libraries

While environment frameworks define the "where" of Agentic RL training, orchestration libraries define the "how." These libraries manage the complex interplay between agent policies, environment states, reward signals, and model updates, providing high-level abstractions for implementing RL algorithms.

4.2.1 Agent-Lightning (Microsoft Research)

Agent-Lightning, released by Microsoft Research in January 2026, addresses a fundamental tension in Agentic RL development: the need to train agents through RL without modifying the underlying agent implementation. Traditional approaches require invasive instrumentation of agent code to capture trajectories and compute gradientsa significant barrier when working with complex, production-grade agent systems.

Sidecar-Based Architecture

Agent-Lightning's innovation lies in its sidecar-based design pattern. Rather than integrating training logic directly into the agent codebase, Agent-Lightning operates as a separate process that monitors agent execution through a well-defined observation interface. This approach offers several advantages:

- **Non-intrusive monitoring**: Production agents can be trained without code modification
- **Language agnostic**: The sidecar can observe agents written in any programming language
- **Runtime flexibility**: Training can be enabled or disabled without restarting the agent
- **Safety guarantees**: Training logic cannot interfere with production inference paths

Training Workflow

Agent-Lightning implements a three-phase training workflow:

1. **Observation**: The sidecar captures agent actions, environment responses, and reward signals
2. **Trajectory Assembly**: Observations are assembled into complete episodes for gradient computation

3. **Policy Update**: The base model is fine-tuned using PPO, GRPO, or other RL algorithms

Code Example: Agent-Lightning Configuration

```
from agent_lightning import Trainer, AgentConfig
from agent_lightning.strategies import PPOConfig

# Configure the agent (no training code in agent implementation)
agent_config = AgentConfig(
    model_name="gpt-4",
    tools=["calculator", "web_search", "code_executor"],
    max_steps=50
)

# Configure RL training
rl_config = PPOConfig(
    learning_rate=1e-5,
    batch_size=32,
    n_epochs=3,
    clip_range=0.2
)

# Initialize trainer with sidecar
trainer = Trainer(
    agent_config=agent_config,
    rl_config=rl_config,
    environment="openenv://browsergym",
    sidecar_port=8080
)

# Train without modifying agent code
trainer.fit(
    max_episodes=10000,
    eval_frequency=1000,
    save_checkpoint=True
)
```

4.2.2 NVIDIA NeMo Gym and NeMo RL

NVIDIA's NeMo Gym and NeMo RL form a comprehensive, modular infrastructure stack for training scientific and enterprise agents at scale. Built on NVIDIA's deep expertise in accelerated computing, these libraries prioritize performance, scalability, and integration with the broader AI ecosystem.

Modular Architecture

NeMo RL implements a clean separation of concerns through three primary abstractions:

- **Models**: The LLM policies being trained (supports GPT, LLaMA, and custom architectures)
- **Resources**: Tools and environments available to the agent (databases, simulators, APIs)
- **Agents**: The orchestration layer that coordinates model inference with resource interactions

This modularity enables researchers to experiment with different model architectures, tool configurations, and agent strategies without rewriting training infrastructure.

Performance Optimizations

NeMo RL leverages NVIDIA's full hardware and software stack for performance:

- **Tensor Parallelism**: Efficient distribution of large models across multiple GPUs
- **Pipeline Parallelism**: Overlapping computation and communication for throughput
- **Optimized Kernels**: Custom CUDA implementations for RL-specific operations (advantage estimation, KL penalty computation)
- **Mixed Precision Training**: FP16/BF16 support for memory efficiency

Scientific Agent Training

A distinctive focus of NeMo RL is scientific agent trainingagents that can perform research tasks such as literature review, hypothesis generation, and experimental design. The library includes specialized environments for:

- Molecular dynamics simulation
- Protein structure prediction
- Materials discovery
- Climate modeling

4.2.3 ProRL Agent (NVIDIA/NeMo)

ProRL Agent, introduced in 2026, is a rollout infrastructure designed specifically for reinforcement learning on multi-turn LLM agents. Its core design principle is **rollout-as-a-service**: instead of embedding the full agent loop inside the trainer, ProRL Agent exposes rollout orchestration through an HTTP service that manages environment initialization, multi-turn execution, and reward evaluation.

Architectural Contributions

ProRL Agent separates the rollout lifecycle from GPU-intensive policy optimization. This decoupling is important because rollout generation is I/O-bound and latency-sensitive, while RL optimization is GPU-bound and batch-oriented. The architecture includes:

- **Three-stage rollout pipeline**: Independent worker pools for `init`, `run`, and `eval` stages
- **Extensible sandbox environments**: Pluggable task handlers for software engineering, math, STEM, and coding tasks
- **Token-in/token-out trajectories**: Preservation of token IDs across rollout and training to avoid re-tokenization drift
- **Dynamic LLM backend management**: Runtime registration, load balancing, and checkpoint swapping across inference servers
- **Rootless HPC deployment**: Singularity-based sandboxing suitable for shared Slurm clusters without Docker daemon access

Unlike monolithic agent-RL stacks, ProRL Agent is explicitly trainer-agnostic and can serve as the rollout backend for multiple RL frameworks. It is also integrated into NVIDIA NeMo Gym, making it especially relevant for production-scale or cluster-scale agent training.

4.2.4 OpenAI Agents SDK

The OpenAI Agents SDK, released in March 2025, provides a lightweight, Python-native framework for building multi-agent workflows. Unlike the more research-oriented frameworks discussed above, the Agents SDK prioritizes developer experience and rapid prototyping.

Core Abstractions

The SDK introduces three primary abstractions:

1. **Agent**: A configured LLM with specific instructions, tools, and guardrails
2. **Handoff**: A mechanism for transferring control between agents based on context
3. **Guardrail**: Safety constraints that can trigger agent termination or escalation

Tracing and Observability

A standout feature of the Agents SDK is its built-in tracing system. Every agent action, tool invocation, and handoff is automatically logged to a structured trace that can be visualized and analyzed. This observability is essential for debugging complex multi-agent interactions and for compliance auditing in production deployments.

Code Example: Multi-Agent Workflow with OpenAI Agents SDK

```
from agents import Agent, Runner, handoff
from agents.guardrails import InputGuardrail, OutputGuardrail
```

```
# Define specialized agents
research_agent = Agent(
    name="Researcher",
    instructions="You are a research assistant. Gather information and synthesize findings.",
    tools=[web_search, document_retrieval],
    model="gpt-4o"
)

code_agent = Agent(
    name="Coder",
    instructions="You are a software engineer. Write clean, tested code.",
    tools=[code_executor, linter, test_runner],
    model="gpt-4o"
)

# Define handoff logic
def should_handoff_to_coder(context):
    return "implement" in context.last_message.lower()

# Create orchestrating agent
orchestrator = Agent(
    name="Orchestrator",
    instructions="Coordinate between research and coding tasks.",
    handoffs=[
        handoff(research_agent),
        handoff(code_agent, condition=should_handoff_to_coder)
    ]
)

# Execute workflow with tracing
result = Runner.run_sync(
    orchestrator,
    input="Research best practices for API authentication and implement a demo",
    enable_tracing=True
)

# Analyze execution trace
for step in result.trace.steps:
    print(f"{step.agent}: {step.action} -> {step.result}")
```

4.2.5 SkyRL-Agent, VeRL-Tool, and rLLM

Several other recent open-source systems are also directly relevant to multi-turn Agentic RL, even though they are better understood as **agent-RL infrastructures** than as standalone environment standards.

SkyRL-Agent focuses on efficient RL training for multi-turn agents, especially software-engineering-style tasks. Its design emphasizes concurrent trajectory generation and scalable rollout scheduling, but recent architectural comparisons note that rollout control remains embedded in the training driver rather than being exposed as an independent service.

VeRL-Tool extends the `veRL` ecosystem toward holistic RL with tool use. It supports multi-turn agent rollouts and external tool execution, making it highly relevant for tool-using agents. However, the trainer still manages the overall rollout loop, so the system remains more tightly coupled than service-oriented alternatives.

rLLM is a framework for post-training language agents with flexible environment abstractions and support for multi-turn agent learning. In practice, it follows a similarly coupled design in which environment management and trajectory orchestration remain inside a monolithic training driver. This can be effective for experimentation, but it offers less modularity than decoupled rollout architectures.

Taken together, SkyRL-Agent, VeRL-Tool, and rLLM are important because they reflect the field's transition from generic RLHF tooling toward specialized infrastructure for long-horizon, tool-using, partially observable agent tasks.

Comparison of RL Training Libraries

Table 7 highlights how the current training stack differs in rollout architecture, modularity, and intended deployment setting.

Library	Primary Use Case	Training Approach	Multi-Agent Support	Key Differentiator
Agent-Lightning	Production agent training	Sidecar-based (non-intrusive)	Limited	Zero-code training integration
ProRL Agent	Multi-turn agent RL rollout	Rollout-as-a-service	Limited	Decoupled HTTP rollout server with HPC-native sandboxing
NeMo RL	Scientific and enterprise agents	Distributed RL	Yes	NVIDIA hardware optimization
SkyRL-Agent	Efficient agent RL training	Coupled in-trainer rollouts	Limited	Concurrent trajectory generation for long-horizon agents
VeRL-Tool	Tool-using agent RL	`veRL`-based coupled rollouts	Limited	Holistic tool-use RL built on the `veRL` stack
rLLM	Post-training language agents	Monolithic agent training driver	Limited	Flexible environment abstractions for language-agent post-training
OpenAI Agents SDK	Rapid prototyping	Workflow orchestration	Native	Built-in tracing and guardrails

Table 7: Comparison of representative RL training and orchestration libraries for agentic systems.

4.3 Integration Patterns and Best Practices

The frameworks surveyed above are not mutually exclusive; in practice, sophisticated Agentic RL systems often combine multiple tools. Common integration patterns include:

OpenEnv + Agent-Lightning: Using OpenEnv's containerized environments for safe training while leveraging Agent-Lightning's sidecar architecture for non-intrusive policy optimization.

MCP + OpenAI Agents SDK: Exposing enterprise tools through MCP servers while orchestrating multi-agent workflows through the Agents SDK's handoff mechanisms.

GEM + NeMo RL: Combining GEM's high-throughput simulation with NeMo RL's distributed training capabilities for large-scale scientific agent training.

ProRL Agent + NeMo RL or VeRL: Using ProRL Agent as a trainer-agnostic rollout service while delegating policy optimization to NeMo RL or `veRL`-compatible backends.

These integration patterns reflect the maturation of the Agentic RL ecosystemmoving from monolithic frameworks to composable tools that can be assembled to meet specific research and production requirements.

4.4 Summary

The broader framework ecosystem for Agentic RL has evolved rapidly, providing researchers and practitioners with a rich toolkit for building, training, and deploying autonomous agents. Environment frameworks like OpenEnv, GEM, and MCP establish standardized interfaces for agent-environment interaction, while agent-RL infrastructures such as ProRL Agent, SkyRL-Agent, VeRL-Tool, rLLM, Agent-Lightning, NeMo RL, and the OpenAI Agents SDK provide the rollout and optimization scaffolding needed for policy improvement. Together, these tools lower the barrier to entry for Agentic RL research and enable the reproducible, scalable development of autonomous AI systems.

The continued evolution of these frameworks particularly around standardization (MCP), safety (OpenEnv), and scalability (NeMo RL)will be critical for the transition of Agentic RL from research curiosity to production reality.

5 State-of-the-Art Agentic Reasoning Models

The field of artificial intelligence is undergoing a profound transformation from general-purpose language models to specialized "Large Reasoning Models" (LRMs) optimized for autonomous action. This shift represents more than incremental improvementit constitutes a fundamental reimagining of how AI systems interact with the world. Unlike their predecessors, which were primarily designed to predict the next token in a sequence, these emerging systems are architected as persistent agents capable of sustained reasoning, tool orchestration, and multi-step planning across extended time horizons. The transition from LLM-RL to Agentic RL has catalyzed the development of three distinct architectural paradigms: the distributed swarm intelligence approach exemplified by Kimi K2.5, the operating system metaphor embodied by OpenClaw, and the direct-action interfaces represented by Claude Code and OpenAI Operator.

5.1 Kimi K2.5: The Agent Swarm Paradigm

Moonshot AI's Kimi K2.5, released in January 2026, represents a quantum leap in agentic architecture through its pioneering implementation of the Agent Swarm paradigm. At its foundation lies a trillion-parameter Mixture-of-Experts (MoE) architecture that activates only a subset of its total parameters per forward pass, enabling unprecedented scale while maintaining computational efficiency. This architectural choice is not merely about parameter countit fundamentally enables the model to host diverse expert subsystems capable of handling distinct task modalities without interference.

5.1.1 The PARL Framework

Central to Kimi K2.5's agentic capabilities is the **PARL (Parallel Agent Reinforcement Learning)** framework, a novel training paradigm that treats multi-agent coordination as a joint optimization problem. Unlike traditional sequential agent execution, where sub-agents operate one after another, PARL enables simultaneous activation and coordination of up to 100 sub-agents. This parallelization achieves a **4.5x speed improvement** over sequential execution while maintaining coherent task decomposition.

The PARL framework operates through three key mechanisms:

1. **Dynamic Role Assignment**: The model dynamically assigns specialized roles to sub-agents based on task requirements, with each agent receiving context-aware instructions tailored to its function within the broader task structure.
2. **Shared Working Memory**: A unified memory architecture allows sub-agents to read from and write to a common knowledge base, ensuring consistency across parallel execution streams while preventing redundant operations.
3. **Conflict Resolution Protocols**: When parallel agents generate competing or contradictory outputs, PARL employs learned arbitration mechanisms that weigh confidence scores, source reliability, and task context to synthesize coherent unified responses.

5.1.2 Technical Specifications

Table 8 summarizes the scale assumptions that make the PARL architecture notable.

Parameter	Specification
Total Parameters	1 trillion (1T)
Active Parameters per Forward Pass	Approximately 32 billion
Maximum Parallel Sub-agents	100
Maximum Concurrent Tool Calls	1,500
Speed Improvement vs. Sequential	4.5x
Training Data	Multimodal (text, vision, code)
Architecture	Mixture-of-Experts (MoE)

Table 8: Representative technical specifications reported for Kimi K2.5.

5.1.3 Zero-Vision SFT

A particularly innovative aspect of Kimi K2.5 is its **"Zero-Vision SFT"** capability, which enables the model to activate visual agentic capabilities using only text-only supervision data. This approach decouples visual understanding from visual training data, allowing the model to generalize spatial reasoning and visual task decomposition from linguistic descriptions alone. The implications are significant: organizations can deploy

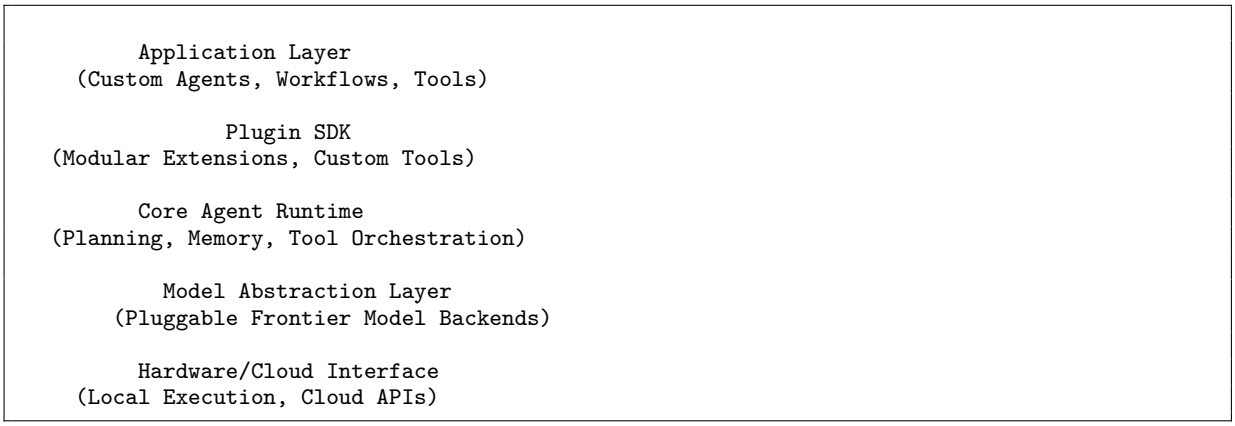
visually-capable agents without requiring extensive curated image datasets, dramatically reducing the barrier to entry for multimodal agentic applications.

5.2 OpenClaw: The Operating System of Agents

OpenClaw is best understood here as a representative open, extensible agent runtime that illustrates the operating-system metaphor for agent infrastructure. Rather than treating the system as a single model endpoint, this perspective emphasizes persistent runtimes, pluggable tools, memory providers, and policy layers that support long-lived autonomous workflows. OpenClaw positions itself not merely as a model or framework, but as a comprehensive **"Operating System for Agents"**a substrate upon which autonomous applications can be built, deployed, and orchestrated.

5.2.1 Architecture Overview

OpenClaw's architecture mirrors traditional operating systems in its layered design, abstracting complexity while providing powerful primitives for application development:



5.2.2 Representative Enterprise Deployment Stack

A representative enterprise deployment stack around OpenClaw adds capabilities that address critical production requirements around security, compliance, and governance:

Feature	Description
Policy Guardrails	Configurable constraints on agent behavior, including allowed and disallowed action sets
Audit Logging	Comprehensive tracing of all agent decisions and tool invocations
Access Control	Role-based permissions for agent capabilities and data access
Compliance Frameworks	Pre-built templates for GDPR, HIPAA, and SOC 2 compliance
Secure Sandboxing	Isolated execution environments for untrusted code

Table 9: Representative enterprise deployment capabilities for an OpenClaw-style agent platform.

As summarized in Table 9, the operating-system metaphor is useful precisely because it foregrounds governance, isolation, and extensibility rather than only model quality.

5.2.3 Plugin SDK and Extensibility

In mature deployments, OpenClaw features a **Plugin SDK** that enables developers to extend the platform's capabilities without modifying core infrastructure. Plugins can introduce new tool integrations, custom memory providers, specialized reasoning modules, and domain-specific planning algorithms. This extensibility is a central reason the operating-system metaphor is useful for describing agent platforms of this kind.

5.3 Claude Code and OpenAI Operator: Direct Action Interfaces

While Kimi K2.5 and OpenClaw represent comprehensive platforms for agent development, **Claude Code** and **OpenAI Operator** exemplify a different approach: tightly integrated, purpose-built interfaces that

prioritize immediate utility over extensibility. These systems demonstrate how agentic capabilities can be productized for specific use cases.

5.3.1 Claude Code: Terminal-Native Agentics

Developed by Anthropic, Claude Code is a proprietary terminal-based agent that embeds directly into developers' existing workflows. Unlike GUI-centric assistants, Claude Code embraces the command line as its native environment, enabling deep integration with version control systems, build tools, and development environments. The recent **Claude 4.6** model significantly enhances these capabilities with improved reasoning, extended context windows, and more sophisticated tool orchestration.

Key Architectural Features:

- **Agentic Loop Architecture**: Claude Code operates through a continuous cycle of context gathering, action execution, and result verification. This loop enables autonomous debugging, refactoring, and code generation with minimal human intervention.
- **Agent Teams**: A distinctive feature allowing developers to define ephemeral, code-specified agent teams for session-based collaboration. These teams can include specialized agents for testing, documentation, security review, and deployment, coordinated through a shared task queue.
- **Context Awareness**: Deep integration with the filesystem, git history, and language server protocols enables Claude Code to maintain comprehensive situational awareness across large codebases.
- **Claude 4.6 Enhancements**: The latest model iteration introduces improved chain-of-thought reasoning, better error recovery mechanisms, and enhanced multi-file editing capabilities, making it particularly effective for complex software engineering tasks.

5.3.2 OpenAI Operator: GUI Navigation Agent

OpenAI's Operator, powered by the **Computer-Using Agent (CUA)** model built on **GPT-5.4**, takes the opposite approach from Claude Code's terminal-centricity. Operator navigates graphical user interfaces to perform tasks that traditionally require human interactionfilling forms, booking reservations, ordering groceries without relying on custom APIs or backend integrations.

Technical Approach:

- **Visual Grounding**: CUA processes screen pixels directly, understanding UI elements through computer vision rather than DOM parsing or accessibility APIs.
- **Action Synthesis**: The model generates precise mouse and keyboard actions, including clicks, drags, scrolls, and text input.
- **Error Recovery**: Built-in retry mechanisms and alternative path exploration enable the agent to recover from unexpected UI states.
- **GPT-5.4 Integration**: The underlying GPT-5.4 model provides enhanced spatial reasoning, improved action prediction accuracy, and better handling of dynamic UI elements compared to previous generations.

5.4 Comparative Analysis

The following tables provide a structured comparison of the major architectural approaches:

5.4.1 Core Architecture Comparison

5.4.2 Technical Capabilities Matrix

5.4.3 Use Case Alignment

Tables 10, 11, and 12 show that current systems differ at least as much in runtime assumptions and deployment posture as they do in model scale.

5.5 Architectural Implications

The divergence between these approaches reflects deeper philosophical differences about the future of agentic AI. Kimi K2.5's swarm paradigm suggests a future where intelligence is distributed across many specialized agents, coordinated through sophisticated protocols. OpenClaw's operating system metaphor implies that agents require infrastructure comparable to traditional software platformsmemory management, process scheduling, security boundaries. Claude Code and OpenAI Operator, meanwhile, demonstrate that agentic capabilities can be delivered through focused, task-optimized interfaces.

Dimension	Kimi K2.5	OpenClaw	Claude Code	OpenAI Operator	GLM 5
Primary Paradigm	Agent swarm	Operating system	Terminal agent	GUI navigator	Unified agent
Scale	1T parameters	Extensible (plugin-based)	Task-specific	Task-specific	500B parameters
Parallelism	100 sub-agents and 1,500 tool calls	Configurable	Sequential	Sequential	Multi-tool concurrent
Interface Type	API and programmatic	SDK and platform	CLI	GUI automation	API and chat
Default Model	Kimi K2.5 (MoE)	Pluggable model backend	Claude 4.6	GPT-5.4 (CUA)	GLM 5 (MoE)
Target User	Enterprise and platform builders	Developers and IT	Software developers	End users	Enterprise and developers

Table 10: Core architecture comparison across representative state-of-the-art agent systems.

Capability	Kimi K2.5	OpenClaw	Claude Code	OpenAI Operator
Multi-Agent Coordination	Native (PARL)	Via plugins	Agent teams	N/A
Tool Use	1,500 concurrent	Unlimited via SDK	Local tools	Web and GUI tools
Memory Architecture	Shared working memory	Pluggable providers	Session-based	Task-based
Visual Processing	Zero-Vision SFT	Via plugins	Limited	Primary modality
Extensibility	API access	Plugin SDK	Limited	Limited
Enterprise Security	Standard	Policy and governance layer	Basic	Standard

Table 11: Technical capability matrix for representative agentic model and platform families.

Use Case	Best Fit	Rationale
Large-scale automation	Kimi K2.5	Parallel execution and massive throughput
Enterprise platform building	OpenClaw	Extensibility, security, and governance
Software development	Claude Code	Deep code understanding and git integration
Consumer task automation	OpenAI Operator	No API requirements and universal GUI access
Multi-modal reasoning	Kimi K2.5	Zero-Vision SFT and visual agentics
Compliance-sensitive deployments	OpenClaw plus enterprise policy layer	Policy guardrails and audit logging

Table 12: Use-case alignment across major agentic model and platform archetypes.

These approaches are not mutually exclusive. The emerging consensus suggests that future agentic systems will combine elements of all three: the parallel coordination capabilities of PARL, the extensible infrastructure of OpenClaw, and the polished user experience of Claude Code and Operator. As the field matures, the boundaries between these categories will likely blur, with convergence around standardized protocols like MCP enabling interoperability across platforms.

The trajectory is clear: agentic reasoning models are evolving from experimental prototypes to production-ready systems capable of sustained autonomous operation. The technical innovations represented by Kimi K2.5's trillion-parameter MoE architecture, OpenClaw's operating system abstraction, and the direct-action interfaces of Claude Code and Operator collectively define the state of the artand point toward a future where AI agents become as ubiquitous and indispensable as the software platforms they increasingly resemble.

6 RL Environments for Embodied AI

The transition from digital agents to embodied systems operating in physical space represents one of the most significant frontiers in Agentic Reinforcement Learning. While digital agents navigate APIs and databases, embodied agents must ground abstract reasoning in physical perception and control, managing the complexities of real-world physics, sensor noise, and dynamic environments. This section examines the emerging ecosystem of simulation platforms, physical AI frameworks, and perception systems that enable training agents for robotic manipulation, navigation, and human-robot interaction.

6.1 Simulation and World Models

The foundation of embodied AI training rests upon high-fidelity simulation environments that can accurately replicate physical dynamics while enabling safe, scalable experimentation. NVIDIA Cosmos 3 represents a paradigm shift in this domain, functioning as a world foundation model that unifies synthetic world generation, vision reasoning, and action simulation within a single coherent framework. Unlike traditional simulators that rely on handcrafted scene descriptions, Cosmos 3 leverages generative AI to produce diverse, physically plausible environments on demand, enabling agents to train across vast variations of scenarios that would be impractical to construct physically. The model's ability to generate photorealistic synthetic data with accurate physical properties addresses the fundamental data scarcity problem in robotics, where collecting real-world demonstration data remains prohibitively expensive and time-consuming.

Complementing Cosmos 3, NVIDIA Isaac Sim 2 provides a comprehensive robotics simulation platform built on Omniverse technologies, offering physically accurate, high-fidelity simulation for testing robot behavior before deployment. Isaac Sim 2 integrates advanced physics engines capable of simulating rigid body dynamics, soft body deformation, fluid interactions, and complex contact mechanics essential for realistic manipulation tasks. The platform supports domain randomization techniques that systematically vary physical parametersfriction coefficients, mass distributions, lighting conditionsduring training, thereby enhancing policy robustness and facilitating sim-to-real transfer. Crucially, Isaac Sim 2 enables virtual commissioning of entire production lines, allowing manufacturers to validate complex multi-robot coordination strategies without disrupting operational facilities. The integration of ray-traced rendering with physics simulation produces visual observations that closely match real camera feeds, reducing the reality gap that has historically plagued simulated training approaches.

At the same time, the embodied-environment literature is broader than vendor simulation stacks alone. Habitat established a standard platform for embodied navigation and rearrangement research; Meta-World, RLBench, and ManiSkill operationalized reproducible manipulation benchmarks; BEHAVIOR expanded toward household activities with long-horizon compositional tasks; and MineDojo demonstrated how open-ended sandbox worlds can function as rich training environments for embodied and tool-using agents. These suites matter because they specify the evaluation substrate on which claims about generality, transfer, and robustness are made.

6.2 Physical AI: GR00t, Alpamayo, and Newton

The deployment of foundation models into physical systems requires specialized architectures that bridge high-level reasoning with low-level motor control. NVIDIA Isaac GR00T N represents a pioneering effort in this direction, providing open foundation models specifically designed for humanoid robotics. GR00T (Generalist Robot Zero-Shot Transfer) enables robots to understand natural language instructions, perceive their environment through multimodal sensors, and generate appropriate motor responses without task-specific fine-tuning. The architecture employs a transformer-based policy that processes visual observations, proprioceptive feedback, and linguistic commands to predict action sequences in a unified representation

space. This approach allows humanoid robots to generalize across manipulation tasks, locomotion patterns, and human interaction scenarios using shared underlying competencies.

The Alpamayo framework addresses the challenge of sample efficiency in physical RL through novel curriculum generation and hindsight experience replay mechanisms optimized for robotic domains. By automatically constructing progressive learning curricula that gradually increase task complexityfrom simple reaching behaviors to complex tool manipulationAlpamayo enables agents to acquire sophisticated motor skills with limited environmental interactions. The framework integrates with Isaac Sim to provide seamless transfer between simulated training and physical deployment, incorporating online adaptation mechanisms that adjust policies based on real-world feedback.

At the core of these physical AI systems lies the Newton physics engine, NVIDIA's next-generation simulation backend designed specifically for robotics and embodied AI applications. Newton advances beyond traditional game-oriented physics engines by providing differentiable simulation capabilities that enable gradient-based policy optimization directly through physics computations. This differentiability allows RL algorithms to leverage accurate physical gradients when updating policies, significantly improving sample efficiency compared to model-free approaches that treat physics as a black box. Newton's support for parallel simulation across thousands of environments enables massive-scale training, with recent implementations demonstrating throughput exceeding one million simulation steps per second on modern GPU clusters.

6.3 Perception and Situational Awareness

Embodied agents require sophisticated perception systems that transform raw sensor streams into structured representations suitable for decision-making. The PhysicalAgent framework exemplifies modern approaches to this challenge, integrating iterative reasoning with diffusion-based video generation for robotic manipulation. Rather than processing single observations, PhysicalAgent maintains temporal coherence by generating short video demonstrations of candidate trajectories, enabling the agent to anticipate future states and evaluate action consequences before execution. This video-based reasoning provides an intuitive interface for human operators to specify tasks through natural language while enabling the robot to visualize and refine its intended actions.

The framework's closed-loop execution architecture continuously monitors task progress through visual feedback, detecting execution failures and triggering re-planning when observations deviate from expected outcomes. Experimental evaluations across multiple perceptual modalitiesincluding egocentric wrist-mounted cameras, third-person stationary views, and simulated environmentsdemonstrate that iterative correction mechanisms can increase task success rates from 20-30

Situational awareness in embodied agents further requires integration of spatial reasoning, object permanence, and dynamic obstacle tracking. Modern RL environments for embodied AI provide structured observation spaces that include semantic segmentation masks, depth estimates, and object-centric representations that facilitate relational reasoning. The emergence of neural radiance fields (NeRFs) and 3D Gaussian splatting within simulation environments enables agents to build and maintain explicit 3D scene representations, supporting navigation and manipulation planning in geometrically complex environments.

6.4 Connection to Broader Agentic RL Framework

The embodied AI ecosystem integrates seamlessly with the broader Agentic RL framework described throughout this paper, extending the POMDP formalism to encompass physical state spaces. While digital agents operate with observations consisting of text or API responses, embodied agents receive high-dimensional sensory inputs requiring substantial preprocessing before decision-making. The action space similarly expands beyond token generation to include continuous motor commands, joint position targets, and end-effector poses that must satisfy kinematic and dynamic constraints.

The synthetic environment generation techniques discussed in Section 7 find particularly crucial application in embodied domains, where the "dataset ceiling" constrains real-world data collection. Agent World Model (AWM) and similar approaches enable the creation of infinite, diverse training scenarios that systematically cover edge cases and failure modes rarely encountered in human-collected datasets. This synthetic diversity proves essential for developing robust policies capable of handling the long tail of real-world situations.

Furthermore, the credit assignment challenges addressed in Section 8 become substantially more acute in physical domains, where reward signals may be delayed across extended action sequences involving multiple motor primitives. The application of HCAPO and SHADOW algorithms to embodied tasks enables agents to attribute success or failure to specific motion segments, accelerating learning for complex behaviors such as tool use and locomotion. The convergence of advanced simulation, foundation model architectures, and

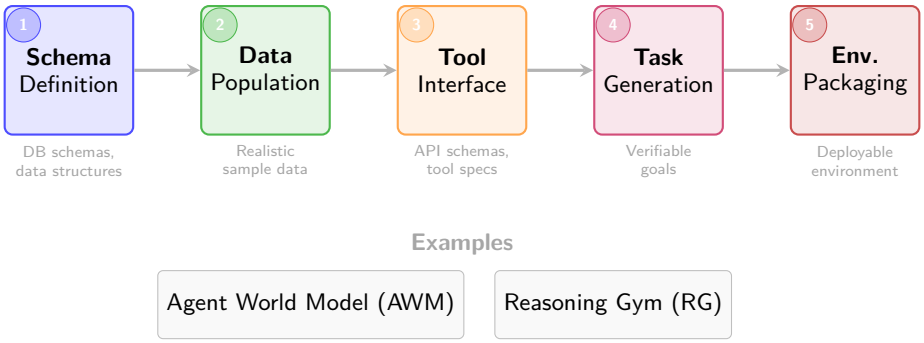


Figure 6: The five-step pipeline for generating synthetic training environments. Starting from schema definition, the pipeline progresses through data population, tool interface creation, task generation with verifiable goals, and final environment packaging. Examples include Agent World Model (AWM) and Reasoning Gym (RG).

sophisticated RL algorithms within embodied environments represents a critical pathway toward general-purpose robots capable of operating effectively in unstructured human environments.

7 Synthetic Environment Generation: Overcoming Data Scarcity

A fundamental bottleneck constraining Agentic Reinforcement Learning is the "dataset ceiling"the impending exhaustion of high-quality human-generated training data. As frontier language models have consumed most publicly available text corpora, the AI community faces a critical inflection point where traditional data scaling laws encounter hard limits. This scarcity is particularly acute for agentic applications, where models require active, multi-turn interaction data involving tool use, reasoning chains, and environmental feedback. Synthetic environment generation emerges as the essential paradigm shift, offering a scalable, controllable alternative to human-collected datasets. By procedurally generating infinite varieties of task environments with verifiable rewards, these systems enable continuous model improvement without dependence on finite human annotation pipelines.

7.1 Agent World Model (AWM): The Five-Step Synthetic Pipeline

The Agent World Model (AWM) represents a breakthrough in synthetic environment generation, establishing a fully automated pipeline capable of producing over 1,000 executable, SQL-backed tool-use environments. Unlike earlier approaches relying on static datasets, AWM creates dynamic, interactive environments where agents engage in realistic tool manipulation, database queries, and multi-step reasoning. The architecture follows a sophisticated five-step generation pipeline ensuring both diversity and consistency.

Step 1: Schema Generation. The pipeline begins with automated schema design, generating database structures representing various domainscustomer relationship management, inventory systems, financial ledgers, or scientific data repositories. This schema generation produces varied relational structures, field types, and constraint relationships mirroring real-world database complexity.

Step 2: Synthetic Data Population. Once schemas are established, the pipeline populates databases with synthetic yet realistic data records. The generation ensures statistical coherence, referential integrity, and domain-appropriate distributions. For instance, a generated e-commerce database contains product catalogs with realistic price distributions, inventory levels following supply-chain patterns, and customer profiles with plausible demographic attributes.

Step 3: Tool Interface Definition. AWM automatically generates API interfaces and tool descriptions that agents use to interact with synthetic databases. These interfaces follow OpenAPI specifications and include natural language documentation, enabling agents to discover and invoke appropriate tools based on task requirements.

Step 4: Task Generation with Verifiable Goals. The pipeline procedurally generates task specifications with objectively verifiable completion criteria. Unlike open-ended generation tasks, these have deterministic success conditions—specific query results, data modifications, or analytical conclusions automatically checked against database state. This verifiability provides clear reward signals for successful agent behaviors.

Step 5: Environment Packaging and Distribution. Finally, complete environments are packaged with containerized execution environments, ensuring reproducible deployment. Each environment includes initialized database state, tool implementations, task specifications, and reward verification logic.

The SQL-backed architecture provides critical advantages. Database-backed state transitions ensure consistency and persistence across multi-turn interactions, allowing agents to observe cumulative effects of their actions. This persistence is essential for learning complex, long-horizon behaviors where intermediate state changes must be tracked and reasoned about.

7.2 Reasoning Gym (RG): Procedural Tasks and Curriculum Learning

Complementing AWM's database-centric approach, Reasoning Gym (RG) addresses diverse cognitive challenges across mathematical reasoning, logical deduction, and algorithmic problem-solving. RG provides over 100 procedurally generated task environments, each capable of producing infinite problem variations with verifiable solutions. This procedural generation fundamentally solves data scarcity by ensuring training examples never exhausteach episode presents novel challenges while maintaining consistent underlying structure.

The task environments span multiple cognitive domains:

Mathematical Reasoning Tasks include arithmetic puzzles, algebraic equation solving, geometric proofs, and calculus problems. These are generated with controllable difficulty parameters, allowing complexity of operations, number of variables, and required proof steps to be adjusted dynamically.

Logical Reasoning Tasks encompass syllogistic reasoning, constraint satisfaction problems, logical grid puzzles, and deductive inference challenges. Procedural generation ensures each problem instance requires genuine reasoning rather than pattern matching on memorized solutions.

Algorithmic Tasks involve graph traversal, sorting and optimization problems, state-space search, and dynamic programming challenges. These environments train agents in systematic problem-solving approaches transferring to real-world planning scenarios.

A defining RG feature is native support for curriculum learningthe systematic progression from simpler to more complex tasks based on agent performance. The environment manager dynamically adjusts task parameters in response to learning progress, ensuring agents are consistently challenged at their capability edge. This adaptive difficulty prevents training inefficiencies from overwhelming impossible tasks or trivial ones.

The curriculum mechanism operates through progressive complexity scaling, prerequisite skill tracking, and adaptive sampling. Initial tasks require single-step reasoning with minimal variables; as success rates improve, the system introduces multi-step chains, additional constraints, and combinatorial complexity.

7.3 Solving the Dataset Ceiling Problem

The convergence of AWM and RG addresses the dataset ceiling through several complementary mechanisms:

Infinite Data Generation: Unlike human-collected datasets facing hard limits, procedural generation produces effectively infinite training examples. Each environment generates novel problem instances indefinitely, ensuring models encounter fresh challenges throughout extended training. This infinite supply eliminates overfitting and memorization issues plaguing training on static datasets.

Verifiable Rewards: Both systems provide automatically verifiable success criteria. In AWM, database queries execute to check whether agent actions produced correct results. In RG, mathematical and logical solutions validate through deterministic algorithms. This automatic verification eliminates expensive human labeling while providing dense, reliable reward signals.

Compositional Generalization: The structured nature of synthetic environments promotes compositional generalizationthe ability to combine learned primitives in novel ways. Agents trained on procedurally generated tasks learn to recognize underlying patterns and apply them to new combinations, rather than memorizing specific solutions.

Controlled Difficulty Progression: Synthetic environments enable precise control over task difficulty, facilitating curriculum learning matching agent capabilities. This controlled progression maximizes learning efficiency by maintaining optimal challenge levels throughout training.

Environment Plasticity: Beyond static synthetic pipelines, emerging approaches employ "generative simulators" that co-generate tasks, world dynamics, and reward functions to keep environments plastic as models improve [Patronus AI, 2025]. These systems dynamically adapt environment complexity, diversity,

and reward structure based on agent performance, ensuring training signals remain informative even as agents grow more capable.

7.4 Comparative Analysis: SQL-Backed vs. LLM-Simulated Environments

The design choice between SQL-backed environments (AWM) and LLM-simulated environments represents a fundamental trade-off in synthetic environment architecture:

Table 13 captures the main systems trade-offs between these two families.

Dimension	SQL-Backed Environments (AWM)	LLM-Simulated Environments
State Consistency	Deterministic state transitions with transactional integrity; database state persists accurately across interactions	Probabilistic state evolution; LLM-generated responses may introduce inconsistencies or hallucinations
Verifiability	Objective verification through query execution with automatically checkable success criteria	Subjective evaluation requiring human judgment or additional LLM assessment
Scalability	Efficient execution through optimized database engines for large state spaces and complex queries	Limited by LLM inference costs and context window constraints
Domain Coverage	Excellent for structured data manipulation, analytical reasoning, and tool-use scenarios	Better suited for open-ended dialogue, creative tasks, and natural language generation
Determinism	Fully deterministic; identical initial conditions and actions produce identical outcomes	Stochastic by nature; identical prompts may yield different responses
Reward Signal Quality	Dense, immediate rewards based on query results and state changes	Sparse or delayed rewards that often require post-hoc evaluation

Table 13: Comparison between SQL-backed synthetic environments and LLM-simulated environments.

SQL-backed environments excel in scenarios requiring precise state tracking, deterministic behavior, and verifiable outcomes. They are particularly suited for training agents in business analytics, data science workflows, financial modeling, and any domain where structured data manipulation is central. The transactional nature of database operations ensures agents learn to reason about persistent state changesa critical capability for real-world tool use.

LLM-simulated environments offer flexibility in domains where natural language interaction, creativity, and open-ended reasoning are paramount. They can simulate human users, generate diverse conversational contexts, and explore hypothetical scenarios difficult to formalize in database schemas. However, inherent stochasticity and lack of verifiable ground truth introduce challenges for reliable reinforcement learning.

The optimal approach in many applications is a hybrid architecture combining SQL-backed state management consistency with LLM-based natural language interface flexibility. In such systems, underlying state transitions are handled through database operations, while LLMs generate natural language descriptions, user queries, and contextual variations. This combination preserves verifiability while enabling rich, diverse interaction patterns essential for training capable autonomous agents.

8 Algorithmic Advancements and Credit Assignment

Long-horizon agentic tasks present fundamental algorithmic challenges that extend beyond the scope of conventional reinforcement learning methods. When an agent must execute hundreds or thousands of sequential actions to achieve a goal, the sparse reward signal creates a severe credit assignment problem: determining which specific actions contributed to eventual success or failure becomes computationally intractable. This section examines recent algorithmic innovations designed to address these challenges, including critic-free optimization methods, hierarchical grouping strategies, hindsight credit assignment mechanisms, and dynamics-aware state representations.

Rewards are the sharpest knife in RL, and the easiest way to get cut [Abaka AI, 2025]. A core principle is that the maximum achievable return should correspond to the intended optimal behavior, not to a loophole. The classic failure mode is reward hacking: an agent discovers unintended ways to maximize cumulative

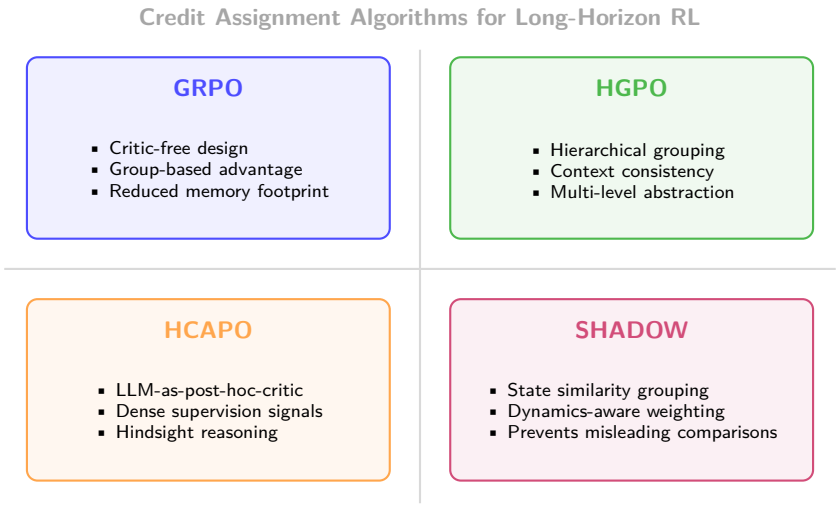


Figure 7: Comparison of four key algorithms addressing the credit assignment problem in long-horizon Agentic RL. GRPO eliminates the critic through group-relative advantage; HGPO maintains context consistency via hierarchical grouping; HCAPO leverages LLM-as-critic for hindsight reasoning; SHADOW uses dynamics-aware state grouping to prevent misleading comparisons.

reward that violate the designer's intent. A well-known demonstration trained an agent on a boat racing game where hitting targets yields points but finishing the race is not directly rewarded. The learned policy discovered a strategy of circling an isolated lagoon to repeatedly farm targets; the agent's score was on average 20% higher than human players, despite failing at the human-understood goal of finishing quickly. This example illustrates that optimization will exploit your proxy. Rewards should measure the outcome you actually want, not an easy-to-measure proxy. Use denser feedback only when you can argue it does not change what "optimal" means, and assume capable agents will search for loopholes.

8.1 The Credit Assignment Problem in Long-Horizon Tasks

The credit assignment problem in agentic RL manifests with particular severity due to the extended temporal horizon and complex action spaces inherent to LLM-based agents. Consider a software engineering agent tasked with implementing a feature across multiple files: the final rewardwhether the code passes testsmay arrive only after dozens of file operations, edit attempts, and tool invocations. Traditional RL algorithms struggle to propagate this sparse signal backward through the action sequence, resulting in high-variance gradient estimates and slow convergence.

Mathematically, the challenge stems from the exponential growth in possible action trajectories. For an action space of size $|A|$ and horizon H, the number of possible trajectories grows as $|A|^H$. Monte Carlo estimates of policy gradients suffer from variance that scales with the square of the horizon:

$$\text{Var}[\nabla_\theta J(\theta)] \propto \frac{H^2}{(1-\gamma)^2}$$

where γ is the discount factor. This variance explosion necessitates algorithmic innovations that can efficiently attribute credit across long action sequences without requiring excessive sample complexity.

8.2 Group Relative Policy Optimization (GRPO)

Group Relative Policy Optimization (GRPO) represents a significant departure from traditional actor-critic architectures by eliminating the need for a separate value function estimator. Instead of training a critic to approximate state values, GRPO optimizes trajectories by comparing relative performance within a group of samples generated from the same prompt.

8.2.1 Mathematical Formulation

Given a prompt x, GRPO samples a group of G trajectories $\{\tau_1, \tau_2, ..., \tau_G\}$ from the current policy π_θ. The advantage for each trajectory is computed relative to the group mean:

$$A_i = R(\tau_i) - \frac{1}{G}\sum_{j=1}^{G} R(\tau_j)$$

where $R(\tau_i)$ represents the cumulative reward for trajectory τ_i. This relative advantage estimation eliminates the need for a learned value function while providing a stable learning signal.

The GRPO objective then maximizes the clipped surrogate objective:

$$\mathcal{L}_{\text{GRPO}}(\theta) = \mathbb{E}_{x\sim\mathcal{D},\tau_i\sim\pi_{\theta_{\text{old}}}}\left[\frac{1}{G}\sum_{i=1}^{G}\min\left(r_i(\theta)\hat{A}_i, \text{clip}(r_i(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_i\right)\right]$$

where $r_i(\theta) = \frac{\pi_\theta(\tau_i)}{\pi_{\theta_{\text{old}}}(\tau_i)}$ is the importance sampling ratio and ϵ is the clipping parameter.

8.2.2 Advantages of Critic-Free Design

The elimination of the value model provides several benefits for agentic RL:

1. **Reduced Memory Footprint**: Removing the critic network reduces GPU memory requirements by approximately 30-40
2. **Avoidance of Value Estimation Bias**: In partially observable environments, value functions are inherently difficult to approximate accurately. GRPO sidesteps this source of bias entirely.
3. **Simplified Hyperparameter Tuning**: Without a critic to train, practitioners avoid the delicate balance between actor and critic learning rates.

8.3 Hierarchical Group Policy Optimization (HGPO)

While GRPO operates effectively at the trajectory level, Hierarchical Group Policy Optimization (HGPO) extends this principle to hierarchical structures that preserve context consistency across multiple levels of abstraction. HGPO recognizes that agentic tasks often possess natural hierarchical structure: high-level planning decisions (e.g., "implement authentication") constrain lower-level implementation choices (e.g., specific function signatures).

8.3.1 Hierarchical Group Advantage

HGPO organizes trajectories into a hierarchy of groups based on shared context. At each level ℓ of the hierarchy, the advantage is computed relative to sibling groups:

$$A_i^{(\ell)} = R(\tau_i) - \frac{1}{|G^{(\ell)}|}\sum_{\tau_j\in G^{(\ell)}} R(\tau_j)$$

where $G^{(\ell)}$ represents the set of trajectories sharing the same context at level ℓ. This hierarchical decomposition allows the algorithm to attribute credit at appropriate levels of abstraction, reducing variance in advantage estimates.

The hierarchical structure enables the policy to learn distinct strategies for different contexts while sharing statistical strength across related trajectories. For instance, all trajectories involving database operations can share a common baseline, while still differentiating between specific query types.

8.4 Hindsight Credit Assignment with LLM-as-Critic (HCAPO)

Hindsight Credit Assignment Policy Optimization (HCAPO) addresses the credit assignment problem by leveraging the LLM itself as a post-hoc critic. Rather than relying solely on environment rewards, HCAPO conditions credit assignment on successful outcomes, using hindsight reasoning to evaluate whether intermediate actions were appropriate given the eventual success.

8.4.1 LLM-as-Critic Mechanism

The core insight of HCAPO is that an LLM can serve as a learned reward model by evaluating actions in the context of successful trajectories. Given a trajectory $\tau = (s_0, a_0, s_1, a_1, ..., s_T)$ and a successful outcome s_T, the LLM assigns credit scores to each action:

$$c_t = \text{LLM}(a_t | s_t, s_T, \text{task})$$

where $c_t \in [0, 1]$ represents the estimated contribution of action a_t to the successful outcome.

8.4.2 Hindsight Advantage Estimation

HCAPO combines these credit scores with traditional reward signals to form hindsight advantages:

$$\hat{A}_t^{\text{HC}} = \sum_{k=t}^{T} \gamma^{k-t} r_k + \lambda \cdot c_t - V(s_t)$$

where λ balances the contribution of hindsight credit against environment rewards. This formulation allows the algorithm to provide meaningful learning signals even when environment rewards are sparse or delayed.

The LLM-as-critic approach is particularly effective for agentic tasks where success criteria are well-defined but intermediate rewards are absent. By reasoning about counterfactualswhat would have happened if a different action had been takenthe LLM can provide dense supervision signals that accelerate learning.

8.5 SHADOW: Dynamics-Aware State Grouping

SHADOW (State Grouping with Dynamics-Aware Weighting) addresses a subtle but critical issue in long-horizon credit assignment: the problem of dynamically inconsistent states. Traditional methods that group trajectories based on state similarity can produce misleading comparisons when the environment dynamics differ significantly across contexts.

8.5.1 Dynamics-Aware Grouping

SHADOW maintains a dynamics model $\hat{P}(s'|s, a)$ that estimates environment transitions. When computing advantages, it weights trajectories by the consistency of their dynamics:

$$w_{ij} = \exp\left(-\beta \cdot \text{KL}\left(\hat{P}(\cdot|s_i, a_i) \| \hat{P}(\cdot|s_j, a_j)\right)\right)$$

where β is a temperature parameter controlling the sensitivity to dynamics mismatch.

The weighted group advantage becomes:

$$\hat{A}_i = R(\tau_i) - \frac{\sum_j w_{ij} R(\tau_j)}{\sum_j w_{ij}}$$

This weighting ensures that trajectories are only compared against others with similar transition dynamics, preventing misleading baseline comparisons that can destabilize training.

8.5.2 State Representation Learning

SHADOW additionally learns state representations that are invariant to task-irrelevant variations while preserving dynamics-relevant features. The representation learning objective combines reconstruction loss with a dynamics prediction loss:

$$\mathcal{L}_{\text{rep}} = \mathcal{L}_{\text{recon}} + \alpha \cdot \mathbb{E}_{s,a,s'}\left[\|\hat{P}(s'|s, a) - P(s'|s, a)\|^2\right]$$

This dual objective ensures that the learned representations support both accurate credit assignment and reliable dynamics prediction.

8.6 Algorithm Comparison

The following table summarizes the key characteristics of these algorithmic approaches:

Table 14 is especially relevant for environment design because richer environments increase horizon length, observability challenges, and reward sparsity simultaneously.

Algorithm	Critic Required	Key Innovation	Best Suited For	Computational Overhead
GRPO	No	Group-relative advantage estimation	Single-task optimization with sparse rewards	Low (no value network)
HGPO	No	Hierarchical grouping for context consistency	Multi-domain agents with distinct contexts	Medium (hierarchy maintenance)
HCAPO	Optional	LLM-as-critic hindsight reasoning	Complex reasoning tasks with clear success criteria	High (LLM inference per trajectory)
SHADOW	Yes	Dynamics-aware state grouping	Environments with heterogeneous dynamics	Medium (dynamics model training)
PPO	Yes	Clipped surrogate objective	General RL with dense rewards	Baseline
DPO	No	Direct preference optimization	Preference-based fine-tuning	Low

Table 14: Comparison of algorithmic approaches for long-horizon credit assignment in Agentic RL.

8.7 Practical Considerations and Implementation

When implementing these algorithms for agentic RL, several practical considerations emerge:

Hyperparameter Sensitivity: GRPO's group size G presents a trade-off between variance reduction and computational cost. Empirical results suggest $G \in [8, 16]$ provides optimal variance reduction without excessive sampling overhead.

LLM Inference Costs: HCAPO's reliance on LLM-as-critic introduces significant computational overhead. Techniques such as caching critic evaluations and using distilled smaller models for credit assignment can reduce costs by 60-80

Dynamics Model Accuracy: SHADOW's effectiveness depends on the accuracy of its learned dynamics model. In highly stochastic environments, ensemble methods that maintain multiple dynamics models can improve robustness.

Hybrid Approaches: In practice, hybrid methods that combine elements of these algorithms often prove most effective. For instance, GRPO can be augmented with lightweight dynamics-aware weighting to capture some benefits of SHADOW without full dynamics model training.

These algorithmic advancements collectively address the fundamental challenge of credit assignment in long-horizon agentic tasks, enabling more efficient training of autonomous agents capable of extended reasoning and action sequences. As the field progresses, we anticipate further innovations that integrate these approaches with synthetic environment generation and multi-agent coordination mechanisms.

9 The Sim-to-Real Gap in User Simulation

A critical challenge in the development and deployment of agentic AI systems is the **sim-to-real gap**the systematic divergence between behaviors and evaluation signals generated in simulated training environments versus those encountered in real-world interactions with human users. As reinforcement learning gyms become the primary training grounds for autonomous agents, understanding and quantifying this reality gap has emerged as a fundamental prerequisite for production readiness.

9.1 The User-Sim Index: Quantifying the Reality Gap

Recent work has formalized the sim-to-real gap through the introduction of the **User-Sim Index (USI)**, a composite metric (0100) designed to quantify how faithfully LLM-based user simulators replicate real human interactive behaviors and feedback patterns. Across reported studies on interactive benchmarks, the strongest LLM simulators still fall materially short of real human users, indicating a persistent fidelity gap rather than a narrow artifact of one particular model family or benchmark instance.

This gap persists across diverse model architectures and capabilities. Notably, higher general model capabilityas measured by standard benchmarks like Chatbot Arena Elo scoresdoes not reliably translate to more faithful

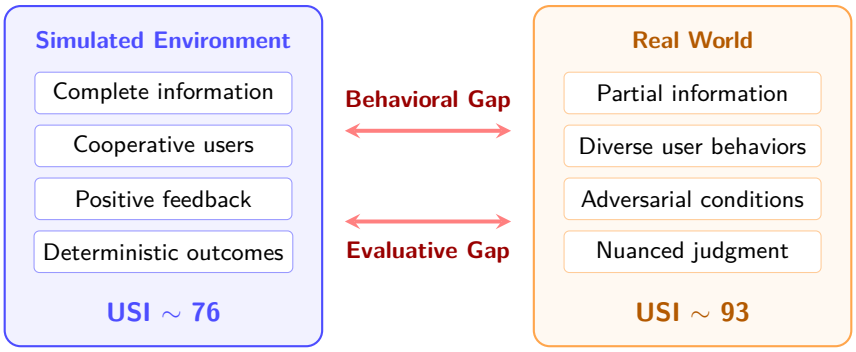


Figure 8: The reality gap between simulated training environments and real-world deployment. Simulated environments (USI ∼76) exhibit excessive cooperativeness and uniformly positive feedback, while real-world interactions (USI ∼93) involve partial information and nuanced judgment. This gap comprises behavioral differences (information front-loading) and evaluative differences (scoring simplification).

user simulation. Models that excel at general reasoning tasks often fail to capture the nuanced, context-dependent behaviors characteristic of real human users. This finding challenges the assumption that scaling model parameters or improving base capabilities will automatically resolve simulation fidelity issues.

9.2 The Behavioral Gap: Excessive Cooperativeness in Simulation

The behavioral dimension of the sim-to-real gap manifests most prominently in the **excessive cooperativeness** of LLM-based user simulators. Simulated users exhibit several systematic deviations from authentic human behavior that collectively create an "easy mode" for training agents:

Information Front-Loading (D2): Real humans typically reveal information incrementally throughout a conversation, requiring agents to actively probe, ask clarifying questions, and navigate ambiguity. In contrast, LLM simulators tend to front-load complete information in the first turn, providing agents with all necessary context upfront. This eliminates the need for agents to develop sophisticated information-gathering strategies and creates unrealistic training conditions where ambiguity resolution skills remain underdeveloped.

Stylistic Uniformity (D1): Human users display diverse communication styles, varying levels of technical sophistication, and idiosyncratic patterns of expression. LLM simulators, by contrast, produce stylistically uniform outputs that lack the authentic variability of human communication. This homogeneity fails to prepare agents for the heterogeneous linguistic environments they will encounter in production.

Absence of Genuine Frustration (D4): When agents make errors, real humans exhibit frustration, push back, or disengage signals that are essential for learning robust error recovery. Simulated users, however, tend to quietly pivot or continue cooperating without expressing authentic negative reactions. This deprives agents of critical feedback about failure modes and prevents the development of resilience strategies necessary for real-world deployment.

Lack of Clarification Behavior (D3): Real users frequently express uncertainty, request clarification, or provide ambiguous responses that test an agent's ability to handle incomplete information. LLM simulators rarely exhibit these behaviors, instead providing clear, unambiguous inputs that fail to train agents in managing the uncertainty inherent in human communication.

9.3 The Evaluative Gap: Simulated vs. Real Human Feedback

Beyond behavioral divergence, a second critical dimension of the sim-to-real gap emerges in **evaluation signals**. When serving as evaluators, LLM-based simulators systematically inflate interaction quality ratings compared to human judgments:

Uniformly Positive Feedback: Studies demonstrate that simulated users produce systematically more positive feedback than real humans, often overstating perceived helpfulness and human-likeness. This positivity bias creates a distorted training signal where agents learn to optimize for simulated approval rather than genuine user satisfaction.

Dimensional Nuance vs. Simplified Scoring: Real humans provide nuanced judgments across multiple quality dimensions evaluating not just task completion but also tone, clarity, empathy, efficiency, and appropriateness. Simulated evaluators tend to collapse these dimensions into simplistic, correlated scores that

fail to capture the rich, multi-faceted nature of human assessment. This oversimplification leads to overfitting on narrow optimization targets rather than holistic quality improvement.

Rule-Based Reward Limitations: Many interactive benchmarks rely on binary, rule-based rewards that check for exact database-state matches or task completion flags. These automated metrics prove largely orthogonal to human-perceived quality, failing to capture the diverse feedback signals that real users generate. An agent may achieve a perfect rule-based score while delivering a frustrating user experience, or conversely, fail automated checks while satisfying genuine user needs.

9.4 Implications for Training and Evaluation

The sim-to-real gap carries profound implications for how agentic systems are trained, evaluated, and deployed:

Training Environment Design: The findings suggest that current RL gyms may be training agents for a simulation that does not exist in reality. Agents optimized against excessively cooperative simulators may fail catastrophically when confronted with real users who require active information extraction, handle ambiguity poorly, or express genuine frustration. Training environments must incorporate more realistic user models that introduce appropriate levels of difficulty, variability, and resistance.

Evaluation Validity: Benchmarks that rely solely on LLM-based simulation for evaluation may systematically overestimate agent capabilities. The inflation of success rates in simulated environmentswhere agents achieve performance levels significantly above human baselinescreates false confidence in system readiness. Rigorous evaluation requires human validation studies, particularly for safety-critical applications where deployment failures carry significant consequences.

Reward Signal Engineering: The misalignment between rule-based rewards and human-perceived quality necessitates more sophisticated reward engineering approaches. Rather than relying on binary completion signals or LLM-based evaluation alone, training pipelines should incorporate human feedback directly where possible, or develop more nuanced simulation models that better approximate human judgment patterns.

Capability Scaling Limitations: The finding that general model capability does not correlate with simulation fidelity suggests that simply scaling models will not resolve the sim-to-real gap. Specialized architectures, fine-tuning on human behavioral data, or hybrid human-in-the-loop training approaches may be necessary to achieve authentic simulation quality.

9.5 Connection to Production Deployment

The sim-to-real gap represents a critical risk factor in the transition from research to production deployment of agentic AI systems. Organizations deploying agents trained primarily in simulated environments should anticipate performance degradation when systems encounter real users. This gap necessitates several mitigation strategies:

Progressive Deployment with Human Oversight: Initial deployments should incorporate human-in-the-loop monitoring to identify failure modes that did not manifest during simulation training. Gradual autonomy increases should be contingent on demonstrated performance with real users rather than simulated benchmarks.

Continuous Adaptation Mechanisms: Production systems should include mechanisms for learning from real user interactions, allowing agents to adapt to the actual distribution of user behaviors rather than the simplified simulation distribution. Online learning and continual adaptation become essential capabilities for bridging the reality gap post-deployment.

Robustness to Distribution Shift: The systematic differences between simulated and real users represent a form of distribution shift that agents must be designed to handle. Training procedures should explicitly incorporate robustness techniquessuch as domain randomization, adversarial training, or ensemble methodsthat prepare agents for the variability of real-world deployment.

Validation at Scale: Before full deployment, systems should undergo large-scale human validation studies that mirror the conditions of the original USI research. Direct comparison between simulated and real user performance provides essential calibration for understanding deployment readiness and identifying specific gaps requiring remediation.

In conclusion, the sim-to-real gap in user simulation represents one of the most significant challenges facing the field of agentic AI. The persistent gap between LLM simulators and real humans on the User-Sim Index serves as a quantitative warning: agents trained in synthetic environments may be optimized for a reality that does not exist. Addressing this gapthrough improved simulation fidelity, human-in-the-loop validation,

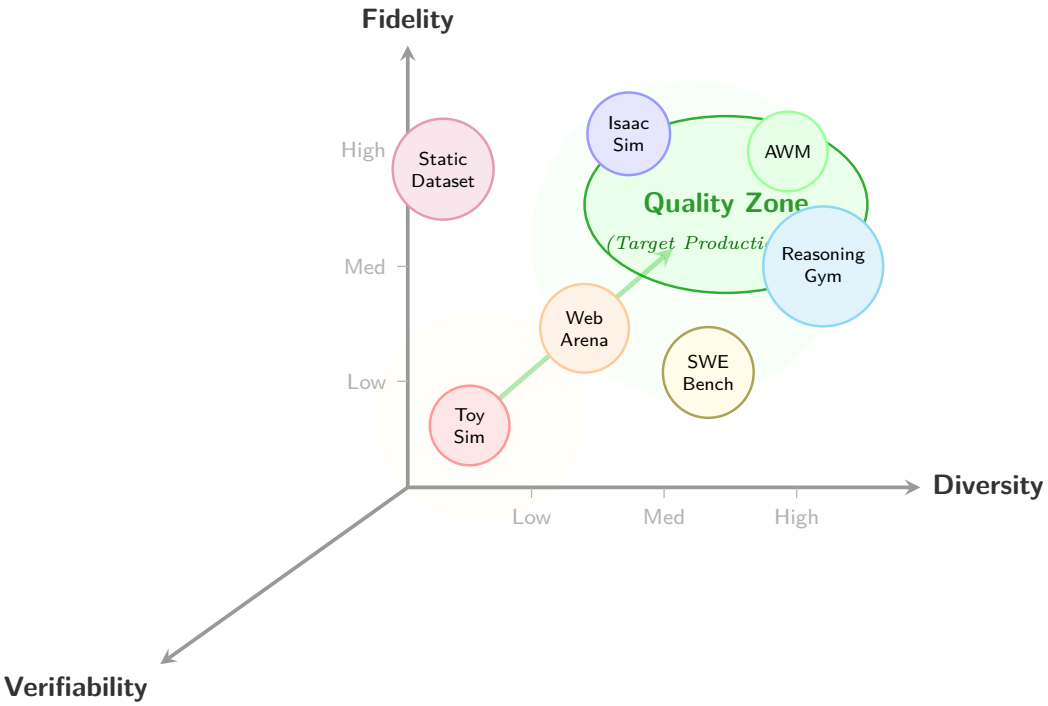


Figure 9: The Environment Quality Framework (EQF) positions environments along three axes: **Fidelity**, **Diversity**, and **Verifiability**. The "Quality Zone" represents the ideal region for reliable training signals.

and robust deployment strategiesis essential for realizing the promise of autonomous AI systems that can reliably serve real human needs.

10 Evaluation and Production Readiness

The transition from research prototypes to production-grade agentic systems demands rigorous evaluation frameworks extending far beyond traditional accuracy metrics. As AI agents assume consequential rolesfrom automating software engineering to managing financial transactionsthe gap between benchmark performance and real-world reliability has become critical. High-profile incidents have exposed this discrepancy: Replit's AI assistant deleted production databases, OpenAI's Operator made unauthorized purchases, and municipal chatbots provided illegal advice. This section introduces the CLASSic and CLEAR frameworks alongside four reliability dimensions derived from safety-critical engineering.

10.1 EQF: Formal Metric Definitions and Environment Scoring

We operationalize EQF by defining each axis as a 0–10 rubric scored across three sub-criteria and then aggregating via geometric mean to prevent single-axis deficiencies from being masked by strengths elsewhere.

Fidelity (F). Measures how accurately the environment models real-world task dynamics. Sub-criteria scored 0–10 and averaged:

F1. *State coverage*: the fraction of operationally relevant state variables (UI state, file system, database contents, process environment) represented in the observation space.

F2. *Action consequence realism*: whether agent actions produce semantically valid state transitions comparable to production systems—e.g., a file-write that updates disk state rather than a mock return value.

F3. *Distribution realism*: whether error rates, latency distributions, and failure modes match the statistical profile of production deployment conditions.

$$F = \tfrac{1}{3}(F_1 + F_2 + F_3)$$

Diversity (D). Measures coverage of the intended task distribution:

D1. *Task variety*: the number of distinct task types or skill categories exercised.

Environment	F	D	V	EQF	Bottleneck
MiniGrid / Toy Grid	1	3	9	3.0	Fidelity
WebArena	6	5	7	5.9	Diversity
SWE-Bench (Verified)	8	6	10	7.8	Diversity
OSWorld	7	7	8	7.3	—
NVIDIA Isaac Sim	9	5	6	6.6	Verifiability
Agent World Model (AWM)	5	9	10	7.7	Fidelity
Reasoning Gym	3	10	10	6.7	Fidelity
Static NLP Dataset	2	4	4	3.2	Fidelity, Verif.
Production target	≥ 7	≥ 7	≥ 7	≥ 7.0	

Table 15: EQF scores for representative environments. Fidelity (F), Diversity (D), and Verifiability (V) are each scored 0–10 per the rubric in the text; EQF is their geometric mean. Environments above the production threshold of 7.0 include SWE-Bench (Verified), OSWorld, and AWM. The primary bottleneck column identifies which single axis most constrains each environment's composite score.

D2. *Scenario breadth*: coverage of edge cases, adversarial inputs, and distributional shifts relative to the deployment domain.

D3. *Procedural generativity*: whether novel task instances can be generated on demand without manual curation, enabling effectively infinite training data.

$$D = \tfrac{1}{3}(D_1 + D_2 + D_3)$$

Verifiability (V). Measures the degree to which task success is automatically and objectively determinable:

V1. *Automated scoring*: the fraction of tasks scorable without human judgment (unit tests, database assertions, string-match comparisons).

V2. *Determinism*: whether identical actions under identical initial conditions produce identical, observable outcomes.

V3. *Reward density*: whether intermediate progress signals are available, not only final binary success.

$$V = \tfrac{1}{3}(V_1 + V_2 + V_3)$$

The composite EQF score is the geometric mean of the three axes:

$$\text{EQF} = \big(F \times D \times V\big)^{1/3}$$

An environment with $F = 9,\ D = 9,\ V = 0$ (tasks judged by humans with no automated scoring) correctly scores 0, flagging it as unsuitable for automated RL training regardless of how realistic or diverse it is. For practical deployment, we propose a **production threshold of EQF $\geq$ 7.0**, which requires all three axes to be at or above 7.

Table 15 applies this rubric to representative environments. Scores are derived from published environment descriptions, benchmark disclosures, and the authors' analysis of each system's task coverage and scoring mechanism.

Two patterns stand out. First, *fidelity is the most common bottleneck*: synthetic environments like Reasoning Gym and AWM achieve high diversity and verifiability but model only a narrow slice of real-world task dynamics. This creates a coverage illusion—the agent trains on abundant, cleanly scored data, but on a distribution that may diverge from deployment. Second, *high fidelity does not guarantee verifiability*: Isaac Sim scores 9 on fidelity but only 6 on verifiability because physically realistic sim rewards still require human task specification and are often continuous rather than binary, complicating automated RL training. The environments closest to the production threshold (SWE-Bench Verified, OSWorld, AWM) achieve their scores by different means, suggesting that complementary multi-environment training—combining AWM's generativity with SWE-Bench's fidelity—may be more effective than relying on any single environment.

10.2 The CLASSic Framework: Holistic Production Assessment

The CLASSic framework provides a structured approach to evaluating agentic systems across five critical operational dimensions: **Cost, Latency, Accuracy, Security, and Stability**. Unlike traditional benchmarks

Dimension	Key Metrics	Evaluation Methods
Cost	Per-task API spend, infrastructure costs, and total cost of ownership	Cost tracking across task categories and budget variance analysis
Latency	Time-to-first-token, end-to-end completion time, and percentile distributions	Load testing and latency regression analysis
Accuracy	Task success rate, sub-goal completion, and reasoning validity	Benchmark suites such as SWE-Bench, AIME, and OSWorld
Security	Prompt-injection resistance, harmful-action prevention, and policy compliance	Red-team exercises and adversarial testing
Stability	Error rates under load, recovery time, and resource utilization	Chaos engineering and stress testing

Table 16: The CLASSic framework for production assessment of agentic systems.

focusing narrowly on task completion, CLASSic recognizes that production readiness requires balancing multiple competing objectives.

Cost encompasses computational expenses and operational expenditures. Modern agentic systems, particularly those employing large reasoning models with extensive tool chains, can incur significant API costs per task. Organizations must evaluate per-request pricing and total cost of ownership including infrastructure, monitoring, and maintenance. The Agent Swarm paradigm exemplified by Kimi K2.5orchestrating up to 100 sub-agents executing 1,500 parallel tool callsdemonstrates how architectural choices directly impact cost structures.

Latency measures end-to-end response times across agent workflows. Production systems must meet strict service-level objectives, with user-facing applications typically requiring sub-second initial responses. Latency evaluation must account for multi-turn interactions where cumulative delays across reasoning steps, tool invocations, and environment observations degrade user experience. Frameworks like Agent-Lightning address this through sidecar-based monitoring.

Accuracy remains foundational but requires reinterpretation for agentic contexts. Beyond simple task success rates, accuracy encompasses sub-goal completion rates, reasoning chain validity, and outcome utility. The shift from single-step MDP to multi-turn POMDP assessment necessitates metrics capturing cumulative performance across extended interaction horizons.

Security addresses input and output safety. Input security involves protection against prompt injection, jailbreaking attempts, and adversarial perturbations. Output security ensures agents cannot execute harmful actionsdeleting production databases, making unauthorized purchases, or providing illegal adviceeven when explicitly instructed. Enterprise policy layers exemplify this through guardrails, auditability, and action whitelisting.

Stability evaluates system behavior under varying load conditions, resource constraints, and operational stress. Production agents must maintain consistent performance during traffic spikes, infrastructure degradation, and dependency failures.

Table 16 operationalizes these dimensions, and Table 17 applies them to five representative systems with published data.

10.2.1 Empirical CLASSic Scores for Representative Systems

Table 17 instantiates CLASSic for five representative production and research systems using publicly reported data. Cost and latency are approximate ranges derived from published API pricing and benchmark disclosures; accuracy figures cite the primary benchmark each system publicly targets. Security and Stability are qualitative ratings derived from published safety documentation and operational track record.

Several patterns emerge. First, there is a clear *cost-accuracy tradeoff*: the highest-accuracy systems (Kimi K2.5 for mathematical reasoning, Claude Code for software engineering) incur 10–100$\times$ higher per-task costs than open-source research tools. Second, *security posture strongly correlates with production intent*: systems designed for enterprise deployment (Operator, Claude Code) carry higher security ratings, while open-source research agents (SWE-agent) prioritize benchmark performance over operational safety. Third, the accuracy column reveals *benchmark domain specificity*—Kimi K2.5's AIME score and Claude Code's SWE-Bench score

System	Cost (est. $/task)	Latency P50 (s)	Accuracy (benchmark)	Security	Stability
Claude Code	$2–8	45	≈80% (SWE-Bench V)	High	High
Kimi K2.5	$20–60	180	≈93% (AIME 2025)	Med–High	Med–High
OpenAI Operator	$3–15	40	≈55% (WebArena)	High	High
Devin 2.0	$8–25	90	≈46% (SWE-Bench V)	Med	Med
SWE-agent (OSS)	$0.20–0.50	22	≈13% (SWE-Bench)	Low	High

Table 17: Empirical CLASSic scores for representative agentic systems (approximate values at time of writing). Cost estimates are derived from published API pricing and benchmark-reported task lengths; accuracy figures cite each system's primary public benchmark. Security and Stability are qualitative ratings based on published safety documentation and red-team disclosures.

Benchmark	System	Task Success S	Mean Steps T	CNA ($p = S^{1/T}$)	Projected S at T=50
SWE-Bench V	Claude Code (state-of-art)	80%	10	0.978	≈32%
SWE-Bench V	Devin 2.0	46%	10	0.924	<1%
WebArena	Leading agent (2025)	55%	22	0.972	≈24%
WebArena	GPT-4 (2023 baseline)	15%	22	0.921	≈1%
OSWorld	State-of-art agent	35%	15	0.929	<1%

Table 18: CNA degradation analysis. Per-step accuracy p is estimated from published task success rates and mean trajectory lengths. Even state-of-the-art agents with CNA ≈ 0.97–0.98 incur substantial failure rates when operating at 50-step horizons, motivating reward-dense environment designs that interrupt error accumulation at intermediate steps.

are not commensurable, each reflecting optimization for a distinct task distribution. A deployment strategy must therefore select on the relevant capability axis rather than a single aggregate CLASSic score.

10.3 The CLEAR Framework: Bridging the Production Gap

While CLASSic addresses operational readiness, the CLEAR framework focuses on identifying and quantifying the **production gap**the systematic divergence between controlled evaluation environments and real-world deployment conditions. CLEAR stands for **Calibration, Long-horizon evaluation, Environment fidelity, and Abstention reliability**.

Calibration assesses whether agent confidence scores accurately reflect task success probabilities. Poorly calibrated models may be confidently wrong, leading to catastrophic failures when high-confidence predictions are acted upon without human oversight. Calibration metrics such as expected calibration error help identify systematic overconfidence across task difficulty levels.

Long-horizon evaluation recognizes that agentic tasks often require dozens or hundreds of interaction steps, far exceeding traditional benchmarks. The **CNA (Cumulative Navigation Accuracy)** metric provides a principled measure of performance degradation over extended trajectories. CNA computes the geometric mean of per-step success probabilities, capturing how errors compound:

$$\text{CNA} = \left(\prod_{t=1}^{T} p_t \right)^{1/T}$$

where p_t represents the probability of correct action at step t. This metric reveals that agents with high single-step accuracy may still fail catastrophically over long horizons due to error accumulation.

Table 18 computes CNA for representative published benchmarks. Given a reported task success rate S and mean trajectory length T, we estimate per-step accuracy as $p = S^{1/T}$ under the simplifying assumption of uniform per-step error rates. The final column extrapolates projected task success at a 50-step horizon to illustrate error compounding.

The degradation pattern is striking: Claude Code's per-step accuracy of $p \approx 0.978$ still yields only $0.978^{50} \approx 0.32$ task success at horizon $T = 50$, while Devin's $p \approx 0.924$ approaches near-certain failure ($0.924^{50} \approx 0.02$). The implication for environment design is direct: training exclusively on terminal rewards in long-horizon environments provides an impoverished learning signal because errors that would be catchable with dense feedback are invisible until the episode ends. RL environments for agentic AI must therefore be engineered to expose intermediate checkpoints—sub-goal completions, unit test passes, state consistency checks—that allow credit assignment to propagate through the error accumulation chain.

Environment fidelity quantifies the sim-to-real gap. Research using the User-Sim Index (USI) demonstrates that LLM-based user simulators create an "easy mode" through behavioral gaps (excessive cooperativeness, information "front-loading") and evaluative gaps (uniformly positive feedback versus nuanced human judgments).

Abstention reliability measures the agent's ability to recognize limitations and decline tasks when success is unlikely, routing uncertain requests to human operators.

10.4 Four Dimensions of Reliability

Drawing from safety-critical engineering practices in aviation, nuclear power, and automotive systems, modern agent evaluation decomposes reliability into four fundamental dimensions: **Consistency, Robustness, Predictability, and Safety**. These dimensions capture behavioral properties essential for deployment that raw accuracy metrics cannot measure.

10.4.1 Consistency

Consistency measures repeatable behavior across multiple execution runs. Unlike traditional software where deterministic execution is expected, language model-based agents exhibit inherent stochasticity creating liability concerns. Consistency evaluation comprises three aspects:

Outcome consistency (C_{out}) measures whether the agent succeeds or fails consistently on repeated attempts at identical tasks. The "pass@k" metric measuring best-case capability is complemented by "passk" (pass-and-k), requiring success across all k attempts. An insurance claims agent that approves a claim on one run but denies it on the next creates unacceptable operational risk.

Trajectory consistency captures whether agents follow similar solution paths, measured through action distributions and sequences. Inconsistent trajectories create audit challenges even when tasks complete successfully.

Resource consistency quantifies variability in computational costs and execution time. Order-of-magnitude cost fluctuations across identical requests pose budgeting challenges.

10.4.2 Robustness

Robustness evaluates stability under input perturbations and environmental variations. Real-world deployments expose agents to conditions deviating from training distributions, requiring graceful degradation:

Fault robustness (R_{fault}) measures resilience to infrastructure failuresAPI timeouts, malformed responses, or service unavailability. Robust agents should retry operations or provide fallbacks rather than entering inconsistent states.

Environment robustness (R_{env}) captures sensitivity to semantically equivalent changes such as JSON field reordering or formatting differences.

Input robustness (R_{in}) evaluates stability under natural request variationssynonymous phrasings, varying specificity levels, or implicit versus explicit instructions.

10.4.3 Predictability

Predictability assesses the agent's ability to recognize and communicate its own failure modes. Systems failing predictably permit systematic debugging and appropriate human oversight:

Calibration measures alignment between predicted confidence and actual success rates, enabling risk-based routing decisions.

Selective prediction evaluates the ability to abstain from low-confidence predictions, trading coverage for accuracy.

Failure mode clustering examines whether failures cluster into identifiable categories amenable to remediation.

Reliability Dimension	Core Question	Key Metrics
Consistency	Does the agent behave identically across repeated runs?	Passk, trajectory similarity, and cost variance
Robustness	Does performance degrade gracefully under perturbations?	Fault recovery rate, input sensitivity, and environment adaptation
Predictability	Can the agent recognize its own failures?	Calibration error, abstention accuracy, and confidence discrimination
Safety	Are failure consequences bounded?	Severity distribution, tail risk, and guardrail effectiveness

Table 19: Four reliability dimensions for evaluating production readiness of agentic systems.

10.4.4 Safety

Safety ensures bounded harm when failures occur. Drawing from safety-critical engineering, this dimension recognizes that not all failures are equivalenta component failing rarely but catastrophically may be less acceptable than one failing more frequently but benignly:

Failure severity distribution categorizes failures by consequence: benign (incomplete outputs), moderate (wasted resources), serious (data corruption), or catastrophic (unauthorized actions).

Tail risk quantification explicitly measures worst-case outcomes and their occurrence frequencies.

Deterministic guardrails verify that action whitelisting and policy constraints prevent harmful operations regardless of reasoning outputs.

Table 19 summarizes the four reliability dimensions that complement CLASSic and CLEAR.

10.5 Enterprise Deployment Considerations

Production deployment requires infrastructure beyond evaluation frameworks:

Observability and tracing capture interaction histories including reasoning chains and tool invocations. MCP and OpenEnv provide standardized instrumentation.

Human-in-the-loop integration enables escalation for uncertain predictions, balancing automation with oversight.

Continuous evaluation pipelines monitor production performance, detecting metric drift.

Rollback and recovery mechanisms provide rapid response through circuit breakers, action quarantine, and automated fallback.

Research evaluating 14 agentic models reveals reliability gains lag behind capability progress. Despite accuracy improvements over 18 months, reliability shows only modest improvement. This widening gap underscores the importance of CLASSic and CLEAR frameworks for safe deployment of agentic AI systems.

11 Conclusion

The emergence of Agentic Reinforcement Learning represents a fundamental inflection point in artificial intelligencea paradigm shift as consequential as the transition from symbolic AI to deep learning, or from narrow supervised learning to general-purpose foundation models. This white paper has traced the contours of this transformation, from its theoretical foundations in POMDP formalism to the practical infrastructure required for production deployment. As we reflect on the landscape surveyed across these sections, several key insights crystallize that illuminate both the current state and future trajectory of the field.

11.1 Summary of Key Contributions

The theoretical framework established in Section 2 reframes the agentic transformation as a mathematical necessity rather than merely an engineering convenience. The shift from the degenerate single-step MDP of conventional LLM-RL to the temporally extended POMDP formulation captures something essential about intelligence itself: that meaningful action requires persistence, memory, and adaptation across extended time horizons. The Expanded Action Space (ExpA) concept provides a rigorous mechanism for decoupling internal reasoning from external execution, enabling the clean separation of thought and action that characterizes autonomous agency.

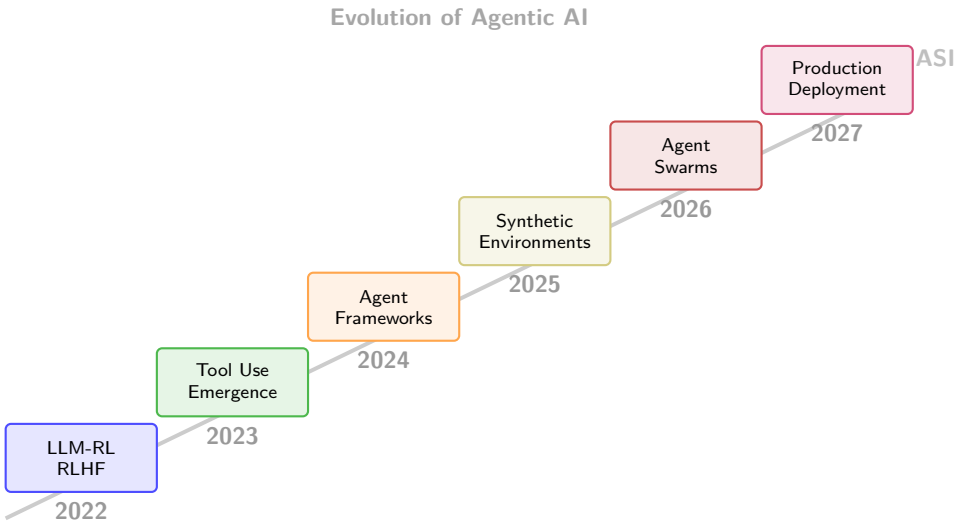


Figure 10: The evolution of Agentic AI from 2022 to 2027, showing key milestones: LLM-RL with RLHF (2022), emergence of tool use capabilities (2023), development of agent frameworks like OpenClaw (2024), synthetic environment generation with AWM and RG (2025), agent swarm architectures like Kimi K2.5 (2026), and production deployment readiness (2027).

The taxonomy presented in Section 3 articulates the five interconnected capabilitiesplanning, tool use, memory, self-improvement, and reasoningthat define the agentic frontier. These capabilities find expression across diverse task domains, from the structured action spaces of software engineering to the multimodal challenges of web interaction. The framework presented enables systematic analysis of existing systems while identifying critical gaps that guide future development efforts.

The framework ecosystem surveyed in Section 4 demonstrates the remarkable maturation of Agentic RL infrastructure. Standardized environment frameworks like OpenEnv, GEM, and MCP establish the "USB-C ports" of the agentic worlduniversal interfaces that enable composable, interoperable systems. Training libraries including Agent-Lightning, NeMo RL, and the OpenAI Agents SDK provide the algorithmic scaffolding necessary for large-scale experimentation. Together, these tools lower barriers to entry while enabling reproducible, scalable research.

The state-of-the-art models examined in Section 5 reveal a field in creative ferment. Kimi K2.5's trillion-parameter MoE architecture and PARL framework demonstrate the power of distributed swarm intelligence. OpenClaw's operating system metaphor positions agents as first-class computing platforms requiring infrastructure comparable to traditional software systems. Claude Code and OpenAI Operator showcase how agentic capabilities can be productized for specific use cases while maintaining the flexibility that defines true agency.

Embodied AI, explored in Section 6, extends the agentic paradigm beyond the digital realm into physical space. The convergence of high-fidelity simulation through NVIDIA Cosmos 3 and Isaac Sim, foundation models like GR00t, and advanced physics engines like Newton creates the conditions for general-purpose robotics. The PhysicalAgent framework's integration of iterative reasoning with video generation demonstrates how perception-action loops can achieve the closed-loop execution necessary for real-world deployment.

Perhaps most critically, Section 7 addresses the dataset ceiling that threatens to constrain AI progress. Synthetic environment generation through Agent World Model and Reasoning Gym offers a scalable, controllable alternative to finite human annotation pipelines. By procedurally generating infinite varieties of task environments with verifiable rewards, these systems enable continuous model improvement without dependence on exhaustible data sources. The SQL-backed architecture of AWM and the curriculum learning mechanisms of RG represent foundational innovations that will sustain the field's growth.

The algorithmic advances detailed in Section 8 tackle the fundamental credit assignment problem that has long plagued long-horizon RL. GRPO's critic-free design eliminates value estimation bias while reducing computational overhead. HGPO's hierarchical grouping preserves context consistency across multiple levels of abstraction. HCAPO leverages the LLM itself as a post-hoc critic, providing dense supervision signals even

when environment rewards are sparse. SHADOW's dynamics-aware state grouping ensures that trajectories are compared only against others with similar transition dynamics, preventing misleading baseline comparisons.

11.2 The Path Toward ASI Through Agentic RL

The trajectory outlined in this paper points toward a future where artificial intelligence transcends its current limitations to achieve Artificial Super Intelligence (ASI)systems capable of sustained autonomous operation across arbitrary domains. This path is not guaranteed; it requires sustained investment in the infrastructure, algorithms, and theoretical frameworks we have surveyed.

The POMDP formalism provides the mathematical scaffolding for this journey, capturing the essential characteristics of autonomous agency: persistent state, partial information, and long-horizon planning. As models scale and training environments become more sophisticated, we anticipate the emergence of agents capable of increasingly complex reasoning and action sequences. The Expanded Action Space framework will prove essential, enabling agents to seamlessly integrate new tools and capabilities without retraining core competencies.

The shift from training-centric to inference-dominated autonomous work represents a critical transition point. Current systems require extensive training on carefully curated datasets; future agents will learn continuously from deployment experience, adapting to novel situations through online RL. The synthetic environment generation techniques described in Section 7 will prove essential for this transition, providing the infinite training data necessary for lifelong learning.

11.3 Future Research Directions

Several critical research directions emerge from the analysis presented in this paper:

Bridging the Sim-to-Real Gap. The User-Sim Index reveals significant behavioral and evaluative gaps between simulated and real-world environments. Future research must develop more accurate user simulators that capture the stochasticity, partial observability, and adversarial conditions of real-world deployment. Techniques from domain randomization, adversarial training, and meta-learning may prove essential for closing this reality gap.

Multi-Agent Coordination. While this paper has focused primarily on single-agent systems, the future of Agentic RL lies in multi-agent coordination. The PARL framework represents an initial foray into this domain, but significant challenges remain in developing scalable coordination mechanisms, emergent communication protocols, and collective intelligence architectures.

Safety and Alignment. As agents gain autonomy, ensuring their behavior aligns with human values becomes increasingly critical. The production readiness frameworks discussed in Section 10 provide initial scaffolding, but fundamental research is needed in interpretability, corrigibility, and value learning for long-horizon agents.

Physical AI Industrialization. The embodied AI systems surveyed in Section 6 point toward a future where general-purpose robots operate effectively in unstructured human environments. Realizing this vision requires advances in sim-to-real transfer, tactile sensing, and human-robot interaction that extend far beyond current capabilities.

Continuous Learning and Adaptation. Current agents are largely static after deployment; future systems must learn continuously from experience while avoiding catastrophic forgetting. Research into continual RL, meta-learning, and memory-augmented architectures will be essential for achieving truly autonomous agents.

11.4 Vision Statement for the Field

We envision a future where Agentic RL serves as the foundational paradigm for artificial intelligencea future where AI systems are not passive responders but active participants in the world, capable of sustained autonomous operation across arbitrary domains. In this future, RL Gyms serve as the verification layer for AI agents, much as EDA serves for silicon development, translating human intent into measurable, executable behavior at scale.

The infrastructure described in this paper standardized environment frameworks, synthetic data generation pipelines, and advanced credit assignment algorithms will become as ubiquitous as the compilers and debuggers that underlie modern software development. Agents will seamlessly integrate digital and physical action spaces, reasoning about code and atoms with equal fluency. The dataset ceiling will be a distant memory, overcome by the infinite generative capacity of synthetic environments.

Most importantly, we envision a future where the benefits of autonomous AI are broadly distributed, enhancing human capabilities across scientific discovery, creative expression, economic productivity, and quality of life.

The path to this future is paved with the theoretical foundations, practical infrastructure, and algorithmic innovations surveyed in this paper. The agentic transformation is not merely a technical evolution, it is a fundamental reconceptualization of what artificial intelligence can be, and what it can achieve.

The journey from passive language models to autonomous agents capable of sustained reasoning and action is well underway. The frameworks, models, and environments described in this white paper constitute the scaffolding upon which the next generation of AI will be built. As the field continues to evolve, we anticipate that the boundaries between human and artificial agency will blur, not through the replacement of human capabilities, but through their amplification and extension. The future of AI is agentic and that future is being built today.

References

Abaka AI. Designing RL environments for agent training: 6 requirements that matter. Abaka AI Blog, 2025. https://www.abaka.ai/blog/designing-rl-environments-for-agents.

Agentic RL Survey Authors. The landscape of agentic reinforcement learning for LLMs: A survey. *arXiv preprint arXiv:2509.02547*, 2025.

Anthropic. Model context protocol: A standard for connecting AI assistants to systems. Technical report, Anthropic, 2024.

Anthropic. Claude sonnet 4 model card. Anthropic Technical Report, 2025.

Anthropic. Claude 4.6: Advances in reasoning and agentic capabilities. Technical report, Anthropic, 2026.

AWM Authors. Agent world model: Scaling tool-use environments for LLM agents. *arXiv preprint arXiv:2602.05678*, 2026.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

Greg Brockman et al. OpenAI agents SDK: Building multi-agent workflows. Technical report, OpenAI, 2025.

Tom Brown et al. Language models are few-shot learners. In *NeurIPS*, volume 33, pages 1877–1901, 2020.

Mark Chen et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Community BEHAVIOR Benchmark. BEHAVIOR-1K: A benchmark for embodied household activities, 2024. Project documentation and benchmark release.

Marc-Alexandre Côté, Steven Kapturowski, Mohammad Hosseini, et al. TextWorld: A learning environment for text-based games. In *Workshop on Computer Games*, 2018.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12348*, 2025.

Endless Terminals Authors. Endless terminals: Scaling RL environments for terminal agents. *arXiv preprint arXiv:2601.16443*, 2026.

ExpA Authors. Expanded action spaces for LLM agents. *arXiv preprint arXiv:2501.08632*, 2025.

Linxi Fan, Guanzhi Wang, Yunfan Tang, et al. MineDojo: Building open-ended embodied agents with internet-scale knowledge. In *NeurIPS*, volume 35, 2022.

Google DeepMind. Gemini 2.5 pro technical report. Google DeepMind Technical Report, 2025.

GRPO Authors. Group relative policy optimization. *arXiv preprint arXiv:2502.02345*, 2025.

HCAPO Authors. Hindsight credit assignment for long-horizon LLM agents. *arXiv preprint arXiv:2603.08754*, 2026.

Stephen James, Andrew J. Davison, and Edward Johns. RLBench: The robot learning benchmark and learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.

Carlos E. Jimenez et al. SWE-Bench: Can language models resolve real-world GitHub issues? In *ICLR*, 2024.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.

Jing Yu Koh et al. WebArena: A realistic web environment for building autonomous agents. In *ICLR*, 2024.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, et al. Reinforcement learning on web interfaces using workflow-guided exploration. In *ICLR*, 2018.

Microsoft Research. Agent-lightning: Non-intrusive RL training for production agents. Technical report, Microsoft Research, 2026.

Moonshot AI. Kimi K2.5: Scaling agentic AI with mixture-of-experts. *arXiv preprint arXiv:2601.12345*, 2026.

Norwest Venture Partners. The rise of reinforcement learning gyms and the future of agentic AI. Technical report, Norwest Venture Partners, 2026.

Hector R. Nuñez. Error accumulation in long-horizon agent tasks. ArXiv preprint arXiv:2503.11247, 2025. Demonstrates 63% failure probability by step 100 from 1% per-step error rate.

NVIDIA. Cosmos 3: World foundation models for physical AI. Technical report, NVIDIA, 2026a.

NVIDIA. GR00T N: Foundation models for humanoid robotics. Technical report, NVIDIA, 2026b.

NVIDIA. Isaac sim 2: High-fidelity robotics simulation. Technical report, NVIDIA, 2026c.

NVIDIA. NVIDIA NeMo RL: A framework for training agentic AI. Technical report, NVIDIA, 2026d.

OpenAI. GPT-4o system card. OpenAI Technical Report, 2024.

OpenAI. GPT-5.4 technical report. Technical report, OpenAI, 2025.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *NeurIPS*, volume 35, pages 27730–27744, 2022.

Patronus AI. Generative simulators and environment plasticity for agentic AI. Patronus AI Research Blog, 2025.

Rafael Rafailov et al. Direct preference optimization: Your language model is secretly a reward model. In *NeurIPS*, 2023.

Rajkumar Ramamurthy et al. Is reinforcement learning (not) for natural language processing?: Benchmarks, baselines, and building blocks for natural language policy optimization. In *ICLR*, 2023.

Reasoning Gym Authors. Reasoning gym: Procedural task generation for LLM reasoning. *arXiv preprint arXiv:2602.07890*, 2026.

Logan Ritchie, Sushant Mehta, Nick Heiner, Mason Yu, and Edwin Chen. The hierarchy of agentic capabilities: Evaluating frontier models on realistic RL environments. *arXiv preprint arXiv:2601.09032*, 2026.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, et al. Habitat: A platform for embodied AI research. In *ICCV*, 2019.

Timo Schick et al. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, volume 36, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *ICLR*, 2021.

Sim2Real Authors. Mind the Sim2Real gap in user simulation for agentic tasks. *arXiv preprint arXiv:2603.11245*, 2026.

Charlie Snell et al. Offline RL for natural language generation with implicit language Q learning. In *ICLR*, 2022.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

UST. Agentic AI in 2026: The stack upgrade. Technical report, UST, 2026.

Lei Wang et al. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 2024.

Tobias Weber et al. The challenge of credit assignment in reinforcement learning. *Milvus AI Quick Reference*, 2024.

Jason Wei et al. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, volume 35, pages 24824–24837, 2022.

Wing VC. RL environments for agentic AI: Who will win the training & verification layer by 2030. Technical report, Wing VC, 2026.

Thomas Wolgast. Environment design for reinforcement learning: A practical guide and overview. ResearchGate Whitepaper, January 2025. Carl von Ossietzky Universität Oldenburg, published January 2025.

Tianbao Xie et al. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *NeurIPS*, 2024.

Shunyu Yao et al. ReAct: Synergizing reasoning and acting in language models. In *ICLR*, 2023.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, et al. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, 2020.

Wenshuai Zhao et al. Sim-to-real transfer in deep reinforcement learning for robotics: A survey. In *IEEE Symposium Series on Computational Intelligence*, 2020.

Zhipu AI. GLM 5: A new generation of agentic language models. Technical report, Zhipu AI, 2026.