Reasoning Is More Than the Model: Harness-Aware Evaluation of Agents on Verifiable Reasoning Tasks

Zhanke Zhou [†1] **Zongze Li** [*1] **Weikai Huang** [*1] **Xuan Li** [*1] **Chentao Cao** [*1] **Xiao Feng** [1] **Xiangyu Lu** [1] **Jinbo Hu** [1] **Menghan Lu** [1] **Yi Xie** [2] **Nico Pelleriti** [3 4] **Shiyang Liu** [1] **Max Zimmer** [3 4] **Brando Miranda** [5] **Jiangchao Yao** [6] **Bo Liu** [2] **Sanmi Koyejo** [5] **Sebastian Pokutta** [3 4] **Bo Han** [1]

Abstract

Open agents are no longer just models: they are model-harness systems whose behavior depends on tool access, control loops, execution feedback, memory, etc. Yet existing evaluations either test models without a harness or fix a single harness to compare models within a domain, obscuring how harness design shapes reasoning capability. To fill this gap, we introduce AlphaDiana, a unified system for harness-aware evaluation of open agents on verifiable reasoning tasks. AlphaDiana standardizes models, harnesses, benchmarks, execution environments, scorers, budgets, and trajectory logging, enabling controlled comparisons across both models and harnesses. We use AlphaDiana to evaluate open agents across mathematical, scientific, coding, and multimodal reasoning tasks, combining macro-level comparisons of direct inference and agentic execution with micro-level ablations of core harness capabilities. Trajectory-level analysis attributes successes and failures to reasoning, tool use, state, budget, and verification, revealing that harnesses can both enable iterative problem-solving and introduce systematic errors. AlphaDiana moves agent evaluation beyond asking whether a system succeeds toward explaining why a model-harness system succeeds or fails. [1]

[†]Team lead [*]Equal contribution [1]TMLR Group, Department of Computer Science, Hong Kong Baptist University [2]Department of Electrical & Computer Engineering, University of Arizona [3]Zuse Institute Berlin [4]Technical University of Berlin [5]Stanford University [6]Cooperative Medianet Innovation Center, Shanghai Jiao Tong University. Correspondence to: Bo Han <bhanml@comp.hkbu.edu.hk>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

[1]Project website: https://alphadiana.org/. Code: https://github.com/tmlr-group/AlphaDiana.

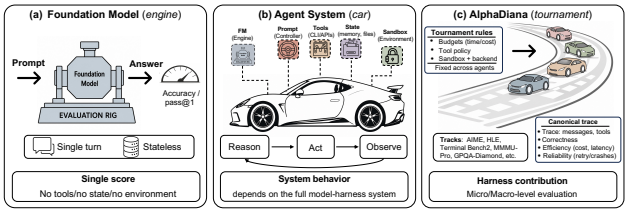


Figure 1. **From model-centric to harness-aware evaluation.** A foundation model is an *engine* evaluated by prompt-in/answer-out scoring; an agent is the *car* that embeds the model in a runtime with control, tools, state, policies, and constraints. AlphaDiana acts as the *tournament organizer*: it fixes evaluation rules across agents, runs them on shared benchmark tracks, and logs canonical traces for analyzing correctness, efficiency, reliability, and failures.

1. Introduction

Open agents such as OpenClaw are increasingly implemented as *model-harness systems*: the model supplies reasoning capacity, while the harness realizes it through tool use, multi-turn control, execution feedback, and memory. This shifts the paradigm from *model* reasoning to *agentic* reasoning. On mathematical and scientific tasks, success often requires not only one-shot generation but also external computation, verification, and iterative refinement (Hendrycks et al., 2021; Cobbe et al., 2021; Rein et al., 2024; Wang et al., 2024; Glazer et al., 2024; Yu et al., 2025b; Jimenez et al., 2024). OpenClaw-like agents make these capabilities operational for problem solving (Lu et al., 2025b; Zhang et al., 2025a; Zhou et al., 2025b; He et al., 2026).

This transition exposes a gap in current evaluation. Existing benchmarks typically either test models without a harness or fix a single harness to compare models within a domain, obscuring how harness design shapes measured reasoning capability (Liang et al., 2022; Suzgun et al., 2022; Liu et al., 2024; Chen et al., 2024; Yao et al., 2024; Kapoor et al., 2026; Rabanser et al., 2026; Bjarnason et al., 2026; He et al., 2026; Lou et al., 2026; Lee et al., 2026). Yet open-agent behavior depends critically on harness-level choices such as tool access and state management, so end-to-end scores conflate base-model reasoning with harness orchestration.

This motivates *harness-aware evaluation*: treating agent performance as a property of the model-harness system rather

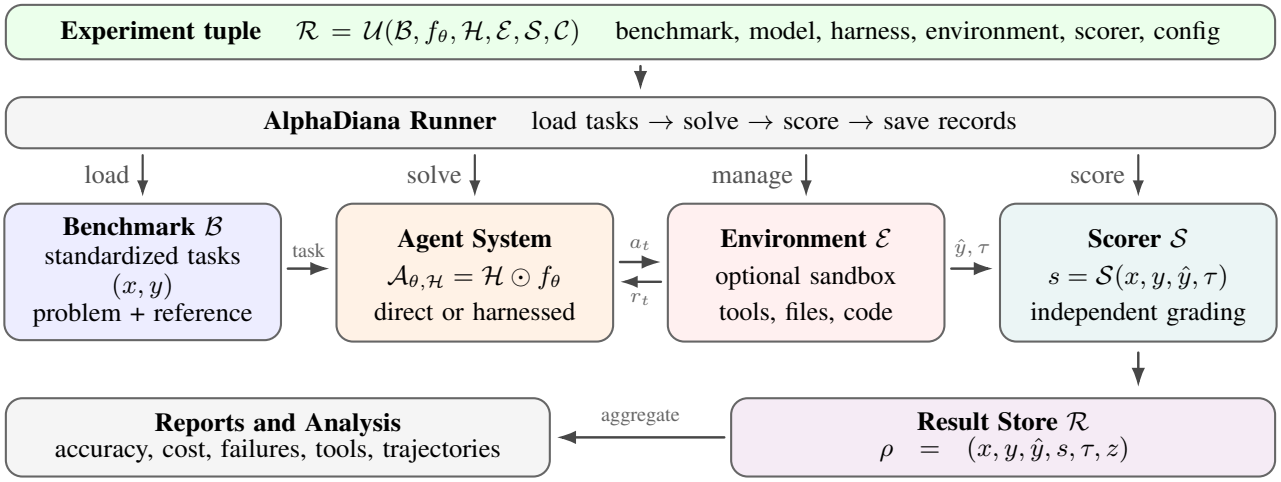


Figure 2. **Architecture of AlphaDiana.** A declarative configuration $\mathcal{C}$ defines the benchmark $\mathcal{B}$, model-harness system $\mathcal{A}_{\theta,\mathcal{H}}$, environment $\mathcal{E}$, scorer $\mathcal{S}$, and budgets; a shared runner $\mathcal{U}$ executes tasks, scores outputs, and saves trajectories in results $\mathcal{R}$ for reproducible analysis.

than the model. We introduce *AlphaDiana*, a unified evaluation system that, as illustrated in Fig. 1, factorizes each run into a benchmark, model, harness, environment, scorer, budget policy, and record. By pairing the same task, scorer, sandbox, and budget with different agents, AlphaDiana turns black-box comparison into controlled experimentation, with logs that enable attribution across reasoning, tool use, execution, recovery, verification, and budget behavior.

Using AlphaDiana, we evaluate OpenClaw-like agents on reasoning tasks across three models (`Qwen3.5-27B`, `Gemma-4-31B-IT`, `Kimi-K2.6`), three harnesses (`OpenClaw`, `ZeroClaw`, `OpenCode`), a no-harness baseline, and four benchmarks (`IMO-AnswerBench`, the verifiable subset of `HLE`, `AIME 26`, `GPQA-Diamond`). We conduct two complementary studies:

- Our **macro-level evaluation** compares direct inference with harnessed execution across matched models, harnesses, and benchmarks. We find that harnessing is neither uniformly beneficial nor model-agnostic: the harness can help one model while hurting another, and the gains and failures arise from model-harness-task interactions.
- Our **micro-level evaluation** asks which harness capabilities explain these changes. By isolating tool access and skill loading, we find non-monotonic effects: tools and skills can improve compatible model-harness settings but distract or destabilize otherwise strong direct reasoning.

Overall, this paper aims to make modern agent evaluation more reproducible, interpretable, and accountable. AlphaDiana does this by assessing reasoning not only as a property of the model, but also as a property of the runtime model-harness system that turns the model into an agent.

2. Related Work

Model-centric reasoning benchmarks. A large body of work evaluates language models on mathematical and scientific reasoning under a prompt-in/answer-out protocol. Representative examples include MATH, FrontierMath, GPQA, SciBench, and HLE (Hendrycks et al., 2021; Glazer et al., 2024; Rein et al., 2024; Wang et al., 2024; Zhai et al., 2026). These benchmarks provide clean task definitions and reliable, often programmatic, verification, making them well-suited to measuring raw *model reasoning*. However, they typically assume a fixed inference protocol and do not distinguish model capability from harness capability in multi-step, tool-using settings.

Interactive agent benchmarks. Benchmarks such as AgentBench, SWE-bench, OSWorld, τ-bench, require tool use, stateful interaction, and long-horizon execution (Liu et al., 2024; Jimenez et al., 2024; Xie et al., 2024; Wu et al., 2025; Lu et al., 2024; Yao et al., 2024; Basu et al., 2024; Merrill et al., 2026; Wang et al., 2025a; Jain et al., 2025). These benchmarks capture realistic agent behavior, but they also create an attribution problem: systems built on the same model may differ substantially because of differences in tool access, control flow, or memory, rather than differences in the model itself. Our work complements this line by asking what end-to-end agent scores measure for a full model-harness agentic system.

Evaluation infrastructure and reliability. Recent work argues that agent evaluation requires stronger infrastructure, standardized protocols, and richer process-level analysis. This includes work on holistic evaluation, reproducible benchmarking, observability, recoverability, and reliability beyond a single scalar metric (Kapoor et al., 2026; Ndzomga, 2026; AlSayyad et al., 2026; Bhardwaj, 2026; Vuddanti & Chittiprolu, 2026; Rabanser et al., 2026; Chen et al., 2024). While related work notes that scaffold design significantly impacts results, we leverage standardization specifically for harness-aware reasoning evaluation. A detailed discussion with more related work can be found in Appendix B.

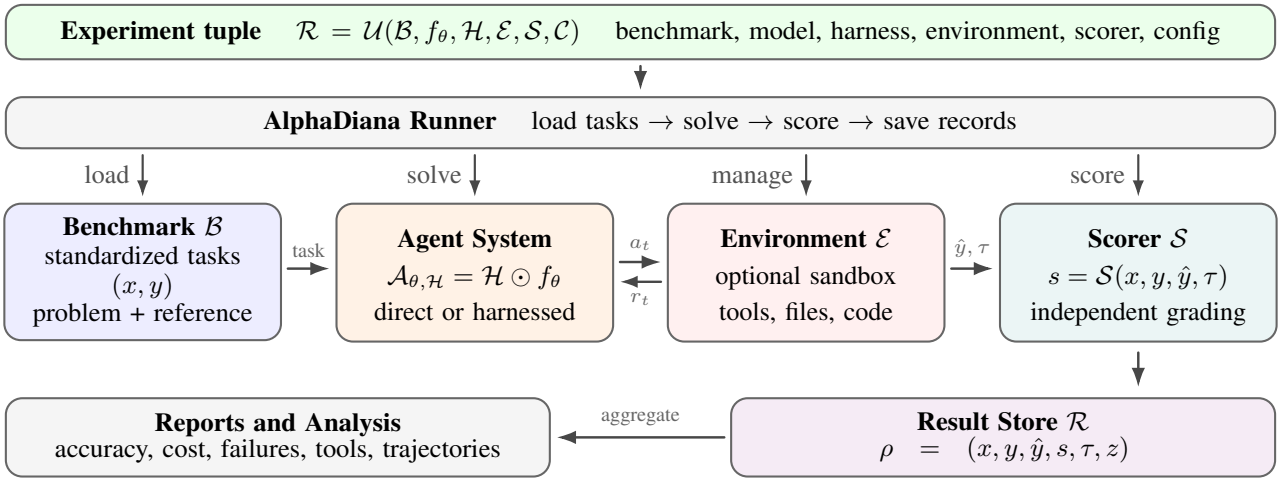


Figure 3. **Architecture of AlphaDiana.** A declarative configuration $\mathcal{C}$ defines the benchmark $\mathcal{B}$, model-harness system $\mathcal{A}_{\theta,\mathcal{H}}$, environment $\mathcal{E}$, scorer $\mathcal{S}$, and budgets; a shared runner $\mathcal{U}$ executes tasks, scores outputs, and saves trajectories in results $\mathcal{R}$ for reproducible harness-aware analysis.

3. AlphaDiana: A Unified System for Evaluating Open Agents

We introduce **AlphaDiana**, a unified system for evaluating open agents as complete runtime model-harness systems. Here, open-agent evaluation poses three challenges beyond standard model benchmarking:

1. **Ambiguous evaluation boundary.** Success or failure may arise from the base model, harness policy, tool interface, execution backend, scorer, or their interaction.
2. **Fragile cross-agent comparison.** Agents may differ in protocols, action spaces, reset rules, budgets, and answer extraction, altering both scores and conclusions.
3. **Stateful and failure-prone execution.** Agent runs are processes that require trajectory preservation, execution isolation, side-effect control, and failure logging.

AlphaDiana addresses these challenges by making the evaluation process explicit, modular, and auditable. As shown in Fig. 3, an experiment run is represented as $\mathcal{R} = \mathcal{U}(\mathcal{B}, f_\theta, \mathcal{H}, \mathcal{E}, \mathcal{S}, \mathcal{C})$, where:

- $\mathcal{R}$ is the collection of saved result records.
- $\mathcal{U}$ is the runner that coordinates task loading, agent execution, scoring, and result persistence.
- $\mathcal{B}$ is the benchmark that defines the task instances.
- f_θ is the base model parameterized by θ.
- $\mathcal{H}$ is the agent harness, which wraps the base model with prompting, tool access, state management, control logic, and answer extraction. We evaluate the agent as the model-harness system $\mathcal{A}_{\theta,\mathcal{H}} = \mathcal{H} \odot f_\theta$; the direct-model baseline is covered by setting $\mathcal{H} = \emptyset$ (no harness).
- $\mathcal{E}$ is the execution environment for tool calls or actions.
- $\mathcal{S}$ is the scorer comparing predictions to task references.
- $\mathcal{C}$ is the run configuration specifying budgets, policies, and logging options.

This formulation standardizes task definition, agent execution, environment interaction, scoring, and result persistence. By separating prompts, normalization, and execution across modules, AlphaDiana pairs the same benchmark, scorer, environment, and budget with different model-harness systems, enabling controlled comparison while preserving trajectories, scores, costs, and failures for auditing.

For each pair of task input and reference object $(x, y) \in \mathcal{B}$, the harness initializes the model prompt p_1 and runtime state m_1. At turn t, the model produces $o_t = f_\theta(p_t)$. If o_t contains a final answer, the harness extracts $\hat{y}$ and terminates; otherwise, it parses an admissible action a_t, dispatches it to the environment $\mathcal{E}$, receives feedback r_t, and updates the prompt to get the next p_{t+1}. Namely,

$$p_t \xrightarrow{f_\theta} o_t \xrightarrow{\mathcal{H}} a_t \xrightarrow{\mathcal{E}} r_t \xrightarrow{\text{PromptUpdate}_{\mathcal{H}}} p_{t+1}.$$

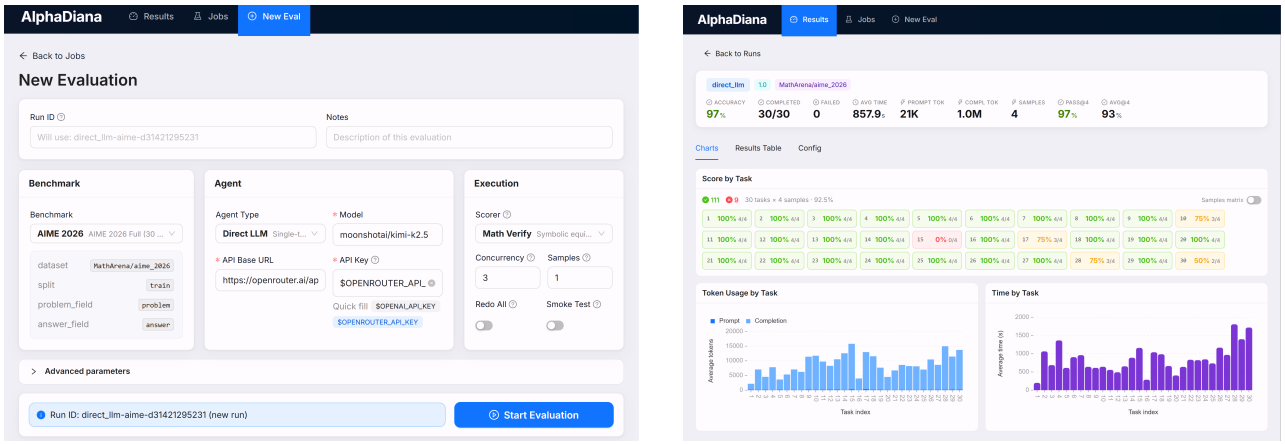
When tool use is required, $\mathcal{E}$ abstracts over execution backends while enforcing isolation, resource limits, reset semantics, and provenance logging. The run stops when the agent returns a final answer, exhausts its budget, or reaches a terminal error state. Then, AlphaDiana records the trajectory,

$$\tau = \Big((p_1, o_1, r_1), \ldots, (p_T, o_T, r_T), \hat{y}\Big),$$

preserving both the prediction and the process that produced it. This is essential for open-agent evaluation, where failures may stem from invalid tool calls, poor feedback recovery, environment errors, or budget exhaustion. After termination, the scorer evaluates the prediction and trajectory:

$$s = \mathcal{S}(x, y, \hat{y}, \tau).$$

The runner $\mathcal{U}$ appends one record per attempt to $\mathcal{R}$, storing the task, prediction, score, trajectory, metadata, resource us-



(a) Create a new evaluation

(b) Obtain the evaluation results

Figure 4. **AlphaDiana dashboard demo. (a)** The run-creation interface instantiates an evaluation from explicit configuration fields. In this example, the benchmark is `AIME 2026`, the agent is `Direct LLM`, and the remaining fields set the model and scorer, which correspond to the run configuration $\mathcal{C}$ and determine which benchmark, model-harness system, scorer, and execution budget are used. **(b)** The dashboard of results $\mathcal{R}$ visualizes an executed `AIME 2026` run with token and time statistics. The summary reports $30/30$ completed tasks, zero failed runs, 97% Pass@4, and 93% Avg@4.

age, and termination status. Preserving both successful and failed attempts enables analyses of accuracy, reliability, cost, and trajectories, including multi-sample aggregates such as pass rate and average correctness. Since all summaries are computed from saved records, reports can be regenerated and audited after completion. Appendix C provides details.

We instantiate this evaluation abstraction in an interactive dashboard, making the configuration and analysis loop explicit and inspectable rather than hidden in scripts. Fig. 4 shows an AIME evaluation workflow: the user creates a run by selecting the benchmark, agent type, model endpoint, scorer, and budget. After execution, the dashboard reports aggregate results and per-task behavior.

4. Macro View: How Open Agents Behave on Verifiable Reasoning Tasks

This section evaluates open agents on verifiable reasoning tasks with the AlphaDiana system. We compare direct inference with harnessed execution to test whether wrapping the model in an agent harness preserves, improves, or degrades measured reasoning capability. Our evaluation covers three open models, `Qwen3.5-27B`, `Gemma-4-31B-IT`, and `Kimi-K2.6`; three harnesses, `OpenClaw`, `ZeroClaw`, and `OpenCode`; and a *Direct* no-harness baseline. We report results on four benchmarks: `IMO-AnswerBench` and `AIME 26` for mathematical reasoning, `GPQA-Diamond` for scientific QA, and the verifiable subset of `HLE` for broad expert-level reasoning. For `AIME 26`, we report Pass@4, the fraction solved by any of four samples, and Avg@4, the mean accuracy across samples. For the other benchmarks, we report Avg@1. In the following, we first analyze final scores as properties of the full evaluations (Sec. 4.1), then analyze trajectories (Sec. 4.2).

4.1. Outcome-based Analysis

Tab. 1 reports the performance of each model-harness system, from which three main findings emerge.

Finding 1. Model Perspective: Harnessing is not a model-agnostic improvement. Its effect depends strongly on model-harness compatibility: the same agent layer can consistently degrade `Qwen3.5-27B`, consistently benefit `Gemma-4-31B-IT`, and yield mixed outcomes for `Kimi-K2.6`.

The same harness can help one model while hurting another. We observe three distinct regimes:

- `Qwen3.5-27B`: Harnesses underperform direct inference on every task. The largest drop occurs on `IMO-AnswerBench`, from 58.3 to 15.8-20.3; on `AIME 26`, Avg@4 falls from 89.2 to 64.2-69.2.
- `Gemma-4-31B-IT`: Harnessing yields consistent gains on `IMO-AnswerBench`, `GPQA-Diamond`, and `AIME 26` across all harnesses; `ZeroClaw` also improves `HLE` from 27.9 to 29.1.
- `Kimi-K2.6`: `ZeroClaw` and `OpenCode` improve `GPQA-Diamond` and `AIME 26`, and `OpenCode` improves `IMO-AnswerBench`. However, `OpenClaw` reduces `GPQA-Diamond` from 77.8 to 31.8.

Finding 2. Task Perspective: Harness benefits are task-dependent. They are most reliable when the task rewards stable computation or answer selection, but become unstable on knowledge-intensive tasks where harness can distract from the model's internal knowledge.

Task structure strongly mediates harness effects, yielding

Table 1. **End-to-end performance of model-harness systems on verifiable reasoning tasks.** Harness rows report absolute metric values, with parenthesized deltas showing absolute differences from the *Direct* baseline. Green/red cells denote improvement/degradation compared to *Direct*.

Model	Harness	IMO-AnswerBench Avg@1	HLE-Verifiable Avg@1	GPQA-Diamond Avg@1	AIME 26	
					Pass@4	Avg@4
Qwen3.5	Direct	58.3	23.0	81.3	96.7	89.2
	OpenClaw	20.3 (-38.0)	13.4 (-9.6)	66.2 (-15.1)	83.3 (-13.4)	64.2 (-25.0)
	ZeroClaw	17.5 (-40.8)	15.0 (-8.0)	77.8 (-3.5)	86.7 (-10.0)	66.7 (-22.5)
	OpenCode	15.8 (-42.5)	13.9 (-9.1)	73.2 (-8.1)	86.7 (-10.0)	69.2 (-20.0)
Gemma-4	Direct	59.0	27.9	83.3	96.7	92.5
	OpenClaw	59.5 (+0.5)	24.2 (-3.7)	85.4 (+2.1)	100.0 (+3.3)	97.5 (+5.0)
	ZeroClaw	61.5 (+2.5)	29.1 (+1.2)	86.4 (+3.1)	100.0 (+3.3)	96.7 (+4.2)
	OpenCode	62.5 (+3.5)	24.0 (-3.9)	87.9 (+4.6)	100.0 (+3.3)	96.7 (+4.2)
Kimi-K2.6	Direct	42.0	35.9	77.8	96.7	85.8
	OpenClaw	27.3 (-14.7)	40.7 (+4.8)	31.8 (-46.0)	93.3 (-3.4)	72.5 (-13.3)
	ZeroClaw	38.7 (-3.3)	33.7 (-2.2)	87.4 (+9.6)	100.0 (+3.3)	93.3 (+7.5)
	OpenCode	48.5 (+6.5)	33.9 (-2.0)	80.8 (+3.0)	100.0 (+3.3)	86.7 (+0.9)

three task-dependent regimes:

- `AIME 26`: Direct inference is already strong; harnesses mainly change reliability, improving Avg@4 for some model-harness pairs but reducing `Qwen3.5-27B`.
- `IMO-AnswerBench`: `Gemma-4-31B-IT` improves consistently, `Kimi-K2.6` only with `OpenCode`, and `Qwen3.5-27B` degrades. Gains appear only when the harness preserves the model's derivation.
- `GPQA-Diamond`/`HLE`: Knowledge-intensive tasks are high-variance: the best `GPQA-Diamond` systems are harnessed, but the largest drop also occurs there; on `HLE`, improvements are rare.

Finding 3. Harness Perspective: No harness is uniformly best. `ZeroClaw` and `OpenCode` provide stable gains across compatible settings, while `OpenClaw` has the widest performance envelope, producing both strong improvements and severe degradations.

We observe that each harness shows a different pattern of gains and failures:

- `ZeroClaw`: Most balanced overall; it improves `Gemma-4-31B-IT` on all axes and gives `Kimi-K2.6` its best `GPQA-Diamond` and `AIME 26` results.
- `OpenCode`: Competitive and strong; it gives `Gemma-4-31B-IT` the best `IMO-AnswerBench` and `GPQA-Diamond` scores, and is the only harness that improves `Kimi-K2.6` on `IMO-AnswerBench`.
- `OpenClaw`: High-variance; it yields the best `HLE` score for `Kimi-K2.6`, but also causes the largest drops on `GPQA-Diamond` and `AIME 26`.

Together, these results motivate harness-aware evaluation: *Direct* is not just a baseline, but a necessary control for separating useful orchestration from harness-induced failures. End-to-end scores should be interpreted as properties of model-harness-task triples, not as standalone measures of model or harness quality. We next turn to trajectories to explain where these gains and degradations arise.

4.2. Process-based Analysis

To explain how model-harness systems produce outcomes, we analyze trajectories at the action level. Following the process of understanding, planning, solving, verifying, and submitting, we assign each step to one of six action types:

- *Understanding*: identifies the goal, givens, constraints, answer choices, and required output format.
- *Planning*: decomposes the task or selects the next strategy.
- *Reasoning*: performs model-internal solving, including derivation, calculation, or comparison.
- *Tool Use*: calls an external tool or incorporates tool feedback into the solution.
- *Verification*: checks, diagnoses, corrects, confirms, or rejects an intermediate result.
- *Finalization*: states the final answer, boxed result, decisive conclusion, or terminal limitation.

This decomposition allows us to move beyond final accuracy and analyze how each agent allocates effort across understanding, planning, reasoning, tool use, verification, and finalization. We apply this action-level analysis to the saved trajectories and obtain the following findings.

Finding 4. Reasoning quality depends on transition control, not only reasoning volume. Across model-harness systems, *Reasoning* often remains the dominant action. The key distinction is whether reasoning transitions toward verification and finalization, or circulates through repeated actions without producing an answer.

In Fig. 5, successful trajectories preserve a structure: the model reasons, checks, uses tools, and finalizes. Failed trajectories contain reasoning, but the reasoning is poorly routed. Instead of resolving uncertainty, it returns to plan-

(a) Qwen-3.5-27B (b) Gemma-4-31B (c) Kimi-K2.6

Figure 5. **Distributions of action w.r.t. the six functional action types.** Deeper color denotes a higher success rate. The black pie chart denotes the ratio of the incorrect trajectories in all trajectories.

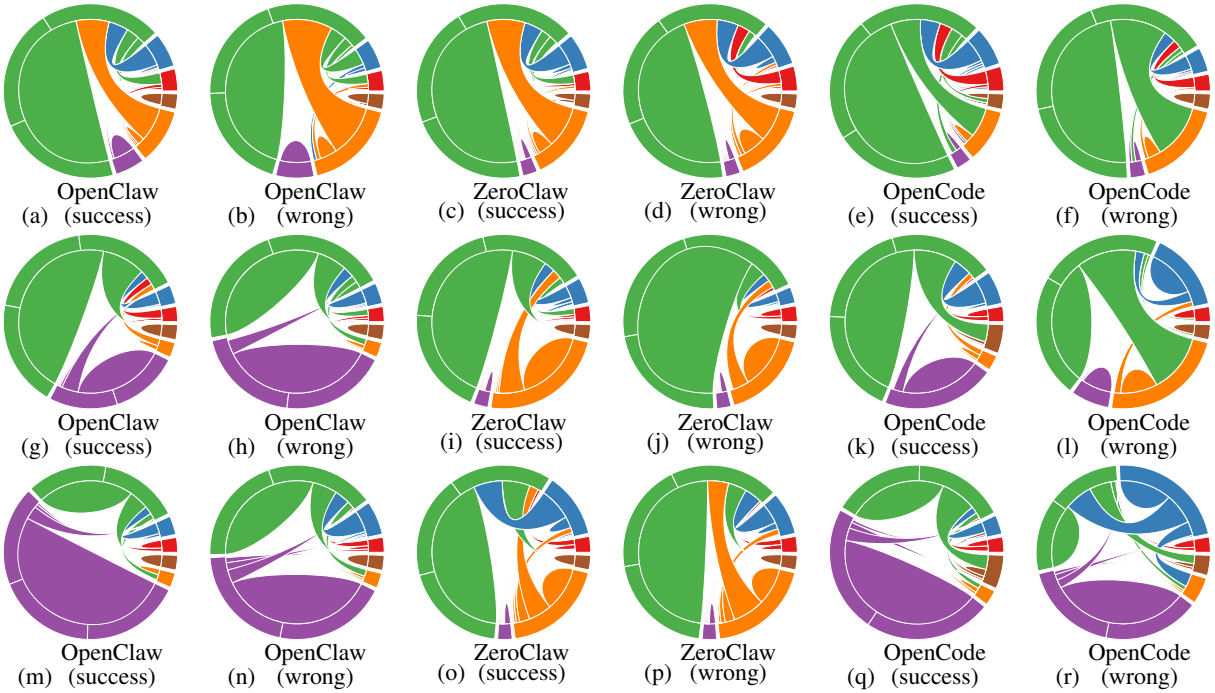


Figure 6. **Action transition chord plot on the HLE dataset.** Colors denote six action categories: *Understanding*, *Planning*, *Reasoning*, *Tool Use*, *Verification*, and *Finalization*. **(a)-(f):** `Qwen3.5-27B`; **(g)-(l):** `Gemma-4-31B-IT`; **(m)-(r):** `Kimi-K2.6`. Chord thickness encodes transition frequency: wider chords indicate more frequent transitions between action pairs. Self-loops denote consecutive occurrences of the same action, outer arcs show each action's total transition mass.

ning, verification, or tool interaction. Thus, failure is not a lack of reasoning but a breakdown in trajectory control.

Finding 5. Harnesses induce distinct failure loops. Different harnesses fail by redirecting the model into different recurrent action patterns: either amplify verification loops or tool-use loops.

Fig. 6 explains why a model can benefit from one harness and degrade under another. A harness is helpful when it inserts useful actions without disrupting the reasoning path. It becomes harmful when the added structure increases action diversity but lengthens the answer path. Action transitions reveal a mechanism: harness quality depends on whether orchestration destabilizes the reasoning-to-answer pathway.

Finding 6. Post-tool entropy reveals whether tool feedback resolves uncertainty. After tool invocation, successful trajectories show an early entropy peak followed by decay, while failed trajectories retain long-tail uncertainty or re-diverge later in the post-tool segment.

In Fig. 7, effective tool use should reduce uncertainty; failed tool use does the opposite. The shape of this failure is harness-dependent: some harnesses produce long uncertainty tails, while others produce early gaps or late spikes.

Finding 7. Confident Long-Tail Failure. Failure is not always marked by high entropy. Wrong trajectories become long and low-entropy, suggesting the model confidently continues after drifting from the correct path.

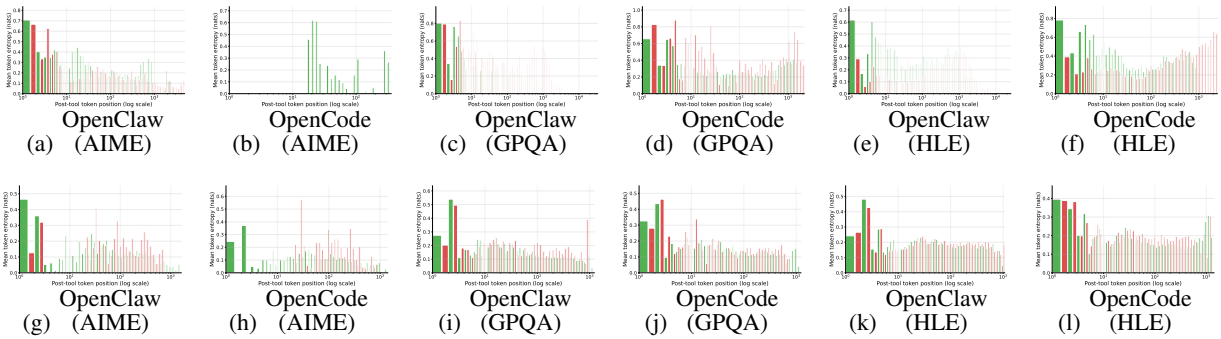


Figure 7. **Post-tool token-wise entropy dynamics after tool invocation.** Bars show the mean entropy of tokens generated after tool calls, indexed by post-tool token position on a log-scaled x-axis. **(a)-(f):** `Qwen3.5-27B`; **(g)-(l):** `Gemma-4-31B-IT`; Green bars denote correct trajectories, and red bars denote wrong trajectories; y-axis emphasizes within-setting entropy dynamics.

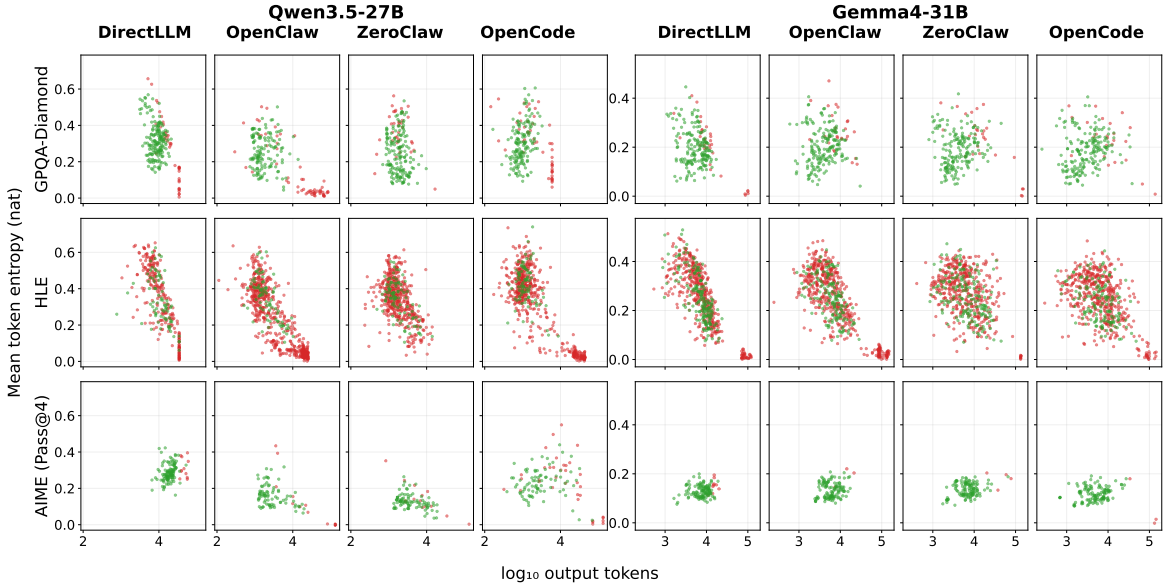


Figure 8. **Trajectory-level mean token entropy versus output token length.** Green points denote correct trajectories, and red points denote incorrect trajectories. Token length is plotted on a log scale.

In Fig. 8, mean entropy alone misses the main signal. Correct trajectories usually stay in a medium-length region with moderate uncertainty, while failures often extend into a long-output tail after entropy has already collapsed. This pattern is strongest in knowledge-intensive settings and weaker on computation-heavy tasks. The key implication is that low entropy should not be read as reliability: in long trajectories, it can indicate commitment to an unproductive path.

5. Micro View: How Harness Capabilities Contribute to Agentic Reasoning

The macro evaluation in Sec. 4 shows that the same harness can help some models while hurting others. We view a harness as a skill set of capabilities, whose marginal effects may vary by model and harness. To isolate these effects, we perform capability-level ablations while holding the model, harness, and benchmark fixed. We study a simple *capability progression* with three harness variants:

$$\textit{Minimal Harness} \longrightarrow \textit{Full Harness} \longrightarrow \textit{Extra Harness}.$$

Minimal Harness. This variant removes tool exposure while keeping the rest of the harness unchanged: $\mathcal{A}_{Minimal} = T_{filter}(\mathcal{H}) \odot f_\theta$. The proxy T_{filter} strips the tool registry and deletes tool descriptions from the system prompt, reducing execution to a chat completion. Comparing $\mathcal{A}_{Minimal}$ with the full harness isolates the marginal effect of tool exposure.

Full Harness. This is the default harness: $\mathcal{A}_0 = \mathcal{H} \odot f_\theta$. It corresponds to the matched model-harness cell in Tab. 1 and serves as the reference configuration for all ablations.

Extra Harness. This variant augments the full harness with a procedural skill library: $\mathcal{A}_{Extra} = S_{load}(\mathcal{H}) \odot f_\theta$. The operator S_{load} adds a pointer to the library while leaving the tools, scorer, and budget unchanged. We evaluate two libraries: a math-specific skill set and a topic-general skill set covering document, UI, and developer workflows without math content. Comparing $\mathcal{A}_{Extra}$ with $\mathcal{A}_0$ tests whether procedural priors help or distract the same system.

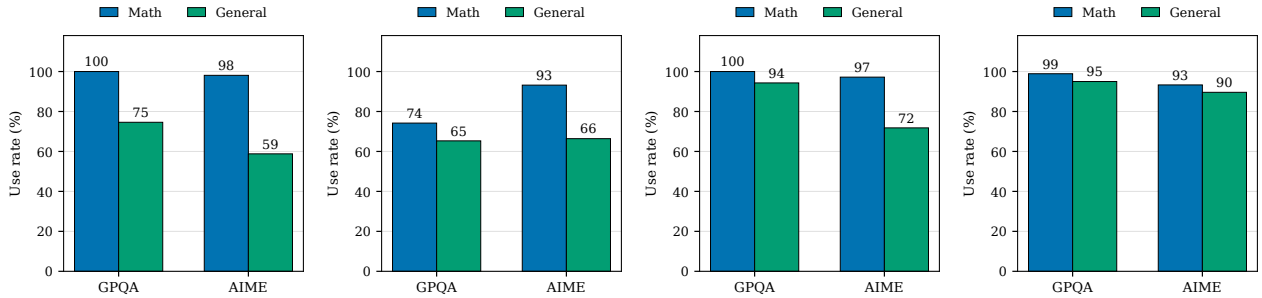
Tabs. 2 and 3 report the per-cell marginal contribution of

Table 2. **Marginal contribution of the tool registry.** 1; *Minimal Harness* applies T_{filter} to strip the tool registry. Parenthesized values are deltas against *Full Harness*.

Model	Harness	GPQA (Avg@1)		AIME 26 (P@4/A@4)	
		Full	Minimal	Full	Minimal
Qwen3.5	ZeroClaw	77.8	77.8 (0.0)	86.7/66.7	86.7/62.5 (-4.2)
	OpenCode	73.2	84.3 (+11.1)	86.7/69.2	93.3/93.3 (+24.1)
Kimi-K2.6	ZeroClaw	87.4	81.3 (-6.1)	100/93.3	96.7/82.5 (-10.8)
	OpenCode	80.8	75.3 (-5.5)	100/86.7	96.7/96.7 (+10.0)

Table 3. **Marginal contribution of loading a procedural skill library.** *Math Skill* and *General Skill* apply S_{load} to load a math-specific and a topic-general library, respectively. Parenthesized values are deltas against *Full Harness*.

Model	Harness	GPQA (Avg@1)			AIME 26 (P@4/A@4)		
		Full	Math	General	Full	Math	General
Qwen3.5	ZeroClaw	77.8	71.8 (-6.0)	80.8 (+3.0)	86.7/66.7	86.7/71.9 (+5.2)	90.9/86.0 (+19.3)
	OpenCode	73.2	71.1 (-2.1)	75.6 (+2.4)	86.7/69.2	76.7/52.5 (-16.7)	80.0/53.3 (-15.9)
Kimi-K2.6	ZeroClaw	87.4	91.6 (+4.2)	91.4 (+4.0)	100/93.3	100/100 (+6.7)	100/100 (+6.7)
	OpenCode	80.8	81.0 (+0.2)	82.1 (+1.3)	100/86.7	100/85.6 (-1.1)	100/87.6 (+0.9)



(a) `ZeroClaw` + `Qwen3.5` (b) `OpenCode` + `Qwen3.5` (c) `ZeroClaw` + `Kimi-K2.6` (d) `OpenCode` + `Kimi-K2.6`

Figure 9. **Skill set use rate per (model, harness) cell.** Fraction of trajectories that read `SKILL.md`, separated by skill library (Math vs. General) and benchmark (GPQA vs. AIME).

the Tool and Skill axes against the matched *Default* configurations from Tab. 1. The Tool axis yields two findings.

Finding 8. Capability contribution is harness-conditioned. Stripping the tool registry degrades `ZeroClaw` on three of four reference cells while improving `OpenCode` on three of four.

T_{filter} strips the tool registry. `ZeroClaw`'s worst drop is -10.8 Avg@4 on `Kimi-K2.6` `AIME 26`; `OpenCode`'s largest gains are $+24.1$ on `Qwen3.5-27B` `AIME 26` and $+11.1$ on `Qwen3.5-27B` GPQA-Diamond. The same intervention helps one harness and hurts the other.

Finding 9. Tools can destabilize reasoning rather than enable it. In `OpenCode`, the tool hurts average accuracy. Removing the tools restores stable success.

On `AIME 26`, `Qwen3.5-27B`'s direct Avg@4 of 89.2 falls to 69.2 inside `OpenCode` (Tab. 1) and recovers to 93.3 once the tool registry is removed, while pass@4 barely shifts. `OpenCode` may let the model get lost; stripping the tools lets it find answers it is already capable of producing.

On the Skill axis, each harness loads either a math-specific (Math) or a general (General) skill set. We examine *whether* it is used (Fig. 9) and *how much* it changes accuracy (Tab. 3).

Finding 10. Models prefer the topic-specific skill set. The Math skill set is consulted more often than the General skill set on every reference cell, and the same model discriminates between the two skill sets by a much larger margin under `ZeroClaw` than under `OpenCode`.

Fig. 9 shows that the model prefers the skill set which matches the task domain. `ZeroClaw` amplifies this: in every cell it reads the Math skill set in over 97% of trajectories, while its General skill set read rate ranges from 59% to 94%. `OpenCode`'s read rates compress to 65–99% across both skill sets, suggesting weaker conditioning on skill-set names or a different read/skip ordering in its agent loop.

Finding 11. Skill loading is mediated by model priors rather than acting as a uniformly helpful add-on. For stronger models, skills induce small and bounded shifts; for weaker models, the same intervention can scaffold reasoning or induce procedural errors.

Tab. 3 shows that across the skill-on cells, `Kimi-K2.6` averages $+2.9$ pp against default with range $[-1.1, +6.7]$, and `Qwen3.5` averages -1.4 pp with range $[-16.7, +19.3]$. `Kimi-K2.6` also has the stronger Full Harness baseline on every reference cell, by 15.3 pp on average. Skill loading is not a model-independent capability injection: its marginal value depends on whether the model can select, adapt, and terminate the procedures exposed by the skill library.

6. Conclusion

We introduced AlphaDiana, a unified system for harness-aware evaluation of open agents on verifiable reasoning tasks. By standardizing models, harnesses, environments, scorers, budgets, and trajectory logging, AlphaDiana enables controlled comparison and post-hoc diagnosis of model-harness systems. Our results show that harnessing is not uniformly beneficial: its effects depend on the model, task, and specific harness capabilities such as tools and skill loading. These findings highlight the need to evaluate agents as complete runtime systems, not models in isolation.

Impact Statement

AlphaDiana makes agent evaluation more rigorous and interpretable by exposing the model-harness boundary and preserving auditable execution traces. A stronger evaluation infrastructure may also accelerate development of more capable autonomous agents, and trajectory logging raises privacy concerns when evaluations involve sensitive prompts or tool outputs. We encourage sandboxed execution, access control, privacy-preserving logging, and transparent reporting of evaluation scope and limitations.

References

Abhyankar, R., Qi, Q., and Zhang, Y. Osworld-human: Benchmarking the efficiency of computer-use agents. *arXiv preprint arXiv:2506.16042*, 2025. URL https://arxiv.org/abs/2506.16042.

Agrawal, L. A., Tan, S., Soylu, D., Ziems, N., Khare, R., Opsahl-Ong, K., Singhvi, A., Shandilya, H., Ryan, M. J., Jiang, M., Potts, C., Sen, K., Dimakis, A. G., Stoica, I., Klein, D., Zaharia, M., and Khattab, O. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025. URL https://arxiv.org/abs/2507.19457.

Almansoori, M., Kumar, K., and Cholakkal, H. Self-evolving multi-agent simulations for realistic clinical interactions. In *MICCAI*, 2025. URL https://arxiv.org/abs/2503.22678.

AlSayyad, A., Huang, K. Y., and Pal, R. Agenttrace: A structured logging framework for agent system observability. *arXiv preprint arXiv:2602.10133*, 2026. URL https://arxiv.org/abs/2602.10133.

Basu, K., Abdelaziz, I., Kate, K., Agarwal, M., Crouse, M., Rizk, Y., Bradford, K., Munawar, A., Kumaravel, S., Goyal, S., Wang, X., Lastras, L. A., and Kapanipathi, P. Nestful: A benchmark for evaluating llms on nested sequences of api calls. *arXiv preprint arXiv:2409.03797*, 2024. URL https://arxiv.org/abs/2409.03797.

Bhardwaj, V. P. Agentassay: Token-efficient regression testing for non-deterministic ai agent workflows. *arXiv preprint arXiv:2603.02601*, 2026. URL https://arxiv.org/abs/2603.02601.

Bjarnason, B. H., Silva, A., and Monperrus, M. On randomness in agentic evals. In *ICLR Workshop on Agents in the Wild*, 2026. URL https://openreview.net/forum?id=UFukKhqCpr.

Buehler, C., Biagiola, M., Grazia, L. D., and Salvaneschi, G. Securing ai agent execution. *arXiv preprint arXiv:2510.21236*, 2025. URL https://arxiv.org/abs/2510.21236.

Bui, N. D. Q. Building effective ai coding agents for the terminal: Scaffolding, harness, context engineering, and lessons learned. *arXiv preprint arXiv:2603.05344*, 2026. URL https://arxiv.org/abs/2603.05344.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A., Leike, J., Achiam, J., Steiner, V., Epstein, Z., van den Driessche, F., Siegler, T., and Sheppard, K. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374.

Chen, Z., Du, W., Zhang, W., Liu, K., Liu, J., Zheng, M., Zhuo, J., Zhang, S., Lin, D., Chen, K., and Zhao, F. T-eval: Evaluating the tool utilization capability of large language models step by step. In *ACL*, 2024. URL https://aclanthology.org/2024.acl-long.515/.

Cheng, D., Huang, S., Gu, Y., Song, H., Chen, G., Dong, L., Zhao, W. X., Wen, J.-R., and Wei, F. Computer environments elicit general agentic intelligence in llms. *arXiv preprint arXiv:2601.16206*, 2026. URL https://arxiv.org/abs/2601.16206.

Chhikara, P., Khant, D., Aryan, S., Singh, T., and Yadav, D. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025. URL https://arxiv.org/abs/2504.19413.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., and Plappert, M. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Coelho, J., Ning, J., He, J., Mao, K., Paladugu, A., Setlur, P., Jin, J., Callan, J., Magalhaes, J., Martins, B., and Xiong, C. Deepresearchgym: A free, transparent, and reproducible evaluation sandbox for deep research. *arXiv preprint arXiv:2505.19253*, 2025. URL https://arxiv.org/abs/2505.19253.

Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., and Su, Y. Mind2web: Towards a generalist agent for the web. In *NeurIPS*, 2023. URL https://arxiv.org/abs/2306.06070.

Drouin, A., Gasse, M., Caccia, M., Laradji, I. H., De Vries, H., Marty, T., Vazquez, D., Chapados, N., and Lacoste, A. Workarena: How capable are web agents at solving common knowledge work tasks? In *ICML*, 2024. URL https://arxiv.org/abs/2403.07718.

Du, M., Xu, B., Zhu, C., Wang, S., Wang, P., Wang, X., and Mao, Z. A-rag: Scaling agentic retrieval-augmented generation via hierarchical retrieval interfaces. *arXiv preprint arXiv:2602.03442*, 2026. URL https://arxiv.org/abs/2602.03442.

Erdogan, L. E., Lee, N., Kim, S., Moon, S., Furuta, H., Anumanchipalli, G., Keutzer, K., and Gholami, A. Plan-and-act: Improving planning of agents for long-horizon tasks. In *ICML*, 2025. URL https://openreview.net/forum?id=ybA4EcMmUZ.

Feng, T., Wang, X., Zhou, Z., Wang, R., Zhan, Y., Li, G., Li, Q., and Zhu, W. Evoagent: Self-evolving agent with continual world model for long-horizon tasks. *arXiv preprint arXiv:2502.05907*, 2025. URL https://arxiv.org/abs/2502.05907.

Ferrag, M. A., Tihanyi, N., and Debbah, M. From llm reasoning to autonomous ai agents: A comprehensive review. *arXiv preprint arXiv:2504.19678*, 2025. URL https://arxiv.org/abs/2504.19678.

Gao, Y., Li, Z., Yuan, Y., Ji, Z., Ma, P., and Wang, S. Skillreducer: Optimizing llm agent skills for token efficiency. *arXiv preprint arXiv:2603.29919*, 2026. URL https://arxiv.org/abs/2603.29919.

Garikaparthi, A., Patwardhan, M., and Cohan, A. Research-gym: Evaluating language model agents on real-world ai research. In *ICLR Workshop on Agents in the Wild*, 2026. URL https://openreview.net/forum?id=eY9RXNbIwr.

Glazer, E., Erdil, E., Besiroglu, T., Chicharro, D., Chen, E., Gunning, A., Olsson, C. F., Denain, J.-S., Ho, A., de Oliveira Santos, E., Järviniemi, O., Barnett, M., Sandler, R., Vrzala, M., Sevilla, J., Ren, Q., Pratt, E., Levine, L., Barkley, G., Stewart, N., Grechuk, B., Grechuk, T., Enugandla, S. V., and Wildon, M. Frontiermath: A benchmark for evaluating advanced mathematical reasoning in AI. *arXiv preprint arXiv:2411.04872*, 2024. URL https://arxiv.org/abs/2411.04872.

Google. Gemini thinking — Gemini API documentation, 2024. URL https://ai.google.dev/gemini-api/docs/thinking.

Guo, D., DeepSeek-AI, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025a. URL https://arxiv.org/abs/2501.12948.

Guo, X., Wang, S., Ji, C., Zhao, X., Xi, W., Liu, Y., Li, Q., Deng, C., and Feng, J. Towards generalized routing: Model and agent orchestration for adaptive and efficient inference. *arXiv preprint arXiv:2509.07571*, 2025b. URL https://arxiv.org/abs/2509.07571.

Gutierrez, B. J., Shu, Y., Gu, Y., Yasunaga, M., and Su, Y. Hipporag: Neurobiologically inspired long-term memory for large language models. In *NeurIPS*, 2024. URL https://openreview.net/forum?id=hkujvAPVsg.

He, C., Zhou, X., Wang, D., Xu, H., Liu, W., and Miao, C. Harness engineering for language agents: The harness layer as control, agency, and runtime. *Preprints.org*, 2026. URL https://www.preprints.org/manuscript/202603.1756.

He, H., Zheng, Z., Wang, W., Wang, Y., Guo, Y., Wang, B., Jin, Y., Wang, X., Zhou, C., Liu, D., et al. Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*, 2024. URL https://arxiv.org/abs/2401.13919.

He, Y., Li, R., Chen, A., Liu, Y., Chen, Y., Sui, Y., Chen, C., Zhu, Y., Luo, L., Yang, F., and Hooi, B. Enabling self-improving agents to learn at test time with human-in-the-loop guidance. In *EMNLP*, 2025. URL https://aclanthology.org/2025.emnlp-industry.115/.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the MATH dataset. *arXiv preprint arXiv:2103.03874*, 2021. URL https://arxiv.org/abs/2103.03874.

Hu, M., Chen, T., Chen, Q., Mu, Y., Shao, W., and Luo, P. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. In *ACL*, 2025. URL https://aclanthology.org/2025.acl-long.1575/.

Hu, Y., Wang, Y., and McAuley, J. Evaluating memory in llm agents via incremental multi-turn interactions. In *ICLR*, 2026. URL https://openreview.net/forum?id=DT7JyQC3MR.

Jain, N., Han, K., Gu, A., Li, W.-D., Yan, F., Zhang, T., Wang, S., Solar-Lezama, A., Sen, K., and Stoica, I. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024. URL https://arxiv.org/abs/2403.07974.

Jain, N., Singh, J., Shetty, M., Zheng, L., Sen, K., and Stoica, I. R2e-gym: Procedural environments and

hybrid verifiers for scaling open-weights swe agents. *arXiv preprint arXiv:2504.07164*, 2025. URL https://arxiv.org/abs/2504.07164.

Jha, S., Paltenghi, M., Maddila, C., Murali, V., Ugare, S., and Chandra, S. Prodcodebench: A production-derived benchmark for evaluating ai coding agents. *arXiv preprint arXiv:2604.01527*, 2026. URL https://arxiv.org/abs/2604.01527.

Jiang, H., Wu, Q., Luo, X., Li, D., Lin, C.-Y., Yang, Y., and Qiu, L. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. In *ACL*, 2024. URL https://aclanthology.org/2024.acl-long.91/.

Jiao, Z., Wang, S., Zhang, Z., Ren, X., Wang, W., Zhao, B., Wei, H., and Zhang, L. Agentic proposing: Enhancing large language model reasoning via compositional skill synthesis. *arXiv preprint arXiv:2602.03279*, 2026. URL https://arxiv.org/abs/2602.03279.

Jiayang, C., Ru, D., Qiu, L., Li, Y., Cao, X., Song, Y., and Cai, X. Amemgym: Interactive memory benchmarking for assistants in long-horizon conversations. In *ICLR*, 2026. URL https://openreview.net/forum?id=sfrVLzsmlf.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. R. Swe-bench: Can language models resolve real-world GitHub issues? In *ICLR*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Kang, M., Chen, W.-N., Han, D., Inan, H. A., Wutschitz, L., Chen, Y., Sim, R., and Rajmohan, S. Acon: Optimizing context compression for long-horizon llm agents. *arXiv preprint arXiv:2510.00615*, 2025. URL https://arxiv.org/abs/2510.00615.

Kapoor, S., Stroebl, B., Kirgis, P., Nadgir, N., Siegel, Z. S., Wei, B., Xue, T., Chen, Z., Chen, F., Utpala, S., Ndzomga, F., Oruganty, D., Luskin, S., Liu, K., Yu, B., Arora, A., Hahm, D., Trivedi, H., Sun, H., Lee, J., Jin, T., Mai, Y., Zhou, Y., Zhu, Y., Bommasani, R., Kang, D., Song, D., Henderson, P., Su, Y., Liang, P., and Narayanan, A. Holistic agent leaderboard: The missing infrastructure for ai agent evaluation. In *ICLR*, 2026. URL https://openreview.net/forum?id=vUaY1t64ZZ.

Kim, D., Ren, Z., Hao, J., Sun, Z., Wang, L., Ma, X., Ye, Z., Han, X., Yin, J., Ji, H., Shen, W., Fan, X., Yao, B., and Guo, C. Beyond perfect apis: A comprehensive evaluation of llm agents under real-world api complexity. *arXiv preprint arXiv:2601.00268*, 2026. URL https://arxiv.org/abs/2601.00268.

Koh, J. Y., Lo, R., Jang, L., Duvvur, V., Lim, M. C., Huang, P.-Y., Neubig, G., Zhou, S., Salakhutdinov, R., and Fried, D. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024. URL https://arxiv.org/abs/2401.13649.

Kumar, L. S., Ba, Y., and Pan, R. Memarchitect: A policy driven memory governance layer. *arXiv preprint arXiv:2603.18330*, 2026. URL https://arxiv.org/abs/2603.18330.

Lee, Y., Nair, R., Zhang, Q., Lee, K., Khattab, O., and Finn, C. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026. URL https://arxiv.org/abs/2603.28052.

Li, H., Zhu, L., Zhang, B., Feng, R., Wang, J., Pan, Y., Barr, E. T., Sarro, F., Chu, Z., and Ye, H. Contextbench: A benchmark for context retrieval in coding agents. *arXiv preprint arXiv:2602.05892*, 2026. URL https://arxiv.org/abs/2602.05892.

Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., Zhang, Y., Narayanan, D., Wu, Y., Kumar, A., et al. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022. URL https://arxiv.org/abs/2211.09110.

Ling, G., Zhong, S., and Huang, R. Agent skills: A data-driven analysis of claude skills for extending large language model functionality. *arXiv preprint arXiv:2602.08004*, 2026. URL https://arxiv.org/abs/2602.08004.

Liu, H., Tian, C., An, N., Wang, Z., Lu, P., Yu, C., and Qi, Q. Budget-constrained agentic large language models: Intention-based planning for costly tool use. *arXiv preprint arXiv:2602.11541*, 2026a. URL https://arxiv.org/abs/2602.11541.

Liu, J., Su, Y., Xia, P., Han, S., Zheng, Z., Xie, C., Ding, M., and Yao, H. Simplemem: Efficient lifelong memory for llm agents. *arXiv preprint arXiv:2601.02553*, 2026b. URL https://arxiv.org/abs/2601.02553.

Liu, S., Yuan, X., Chen, T., Zhan, Z., Han, Z., Zheng, D., Zhang, W., and Cao, S. Caster: Breaking the cost-performance barrier in multi-agent orchestration via context-aware strategy for task efficient routing. *arXiv preprint arXiv:2601.19793*, 2026c. URL https://arxiv.org/abs/2601.19793.

Liu, X., Yu, H., Xu, H., Xu, Y., Lei, Z., Lai, H., Yu, Y., Feng, Y., Tang, Z., Meng, Y., et al. Agentbench: Evaluating LLMs as agents. In *ICLR*, 2024. URL https://arxiv.org/pdf/2308.03688.

Lou, X., Lazaro-Gredilla, M., Dedieu, A., Wendelken, C., Lehrach, W., and Murphy, K. P. Autoharness: improving llm agents by automatically synthesizing a code harness. *arXiv preprint arXiv:2603.03329*, 2026. URL https://arxiv.org/abs/2603.03329.

Lu, J., Holleis, T., Zhang, Y., Aumayer, B., Nan, F., Bai, F., Ma, S., Ma, S., Li, M., Yin, G., Wang, Z., and Pang, R. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. *arXiv preprint arXiv:2408.04682*, 2024. URL https://arxiv.org/abs/2408.04682.

Lu, M., Sun, W., Du, W., Ling, Z., Yao, X., Liu, K., and Chen, J. Scaling llm multi-turn rl with end-to-end summarization-based context management. *arXiv preprint arXiv:2510.06727*, 2025a. URL https://arxiv.org/abs/2510.06727.

Lu, M., Wu, M., Liu, F., Xu, J., Li, W., Wang, H., Hu, Z., Ding, Y., Sun, Y., Lu, J., and Zhang, Y. Choosing how to remember: Adaptive memory structures for llm agents. *arXiv preprint arXiv:2602.14038*, 2026. URL https://arxiv.org/abs/2602.14038.

Lu, P., Chen, B., Liu, S., Thapa, R., Boen, J., and Zou, J. Octotools: An agentic framework with extensible tools for complex reasoning. *arXiv preprint arXiv:2502.11271*, 2025b. URL https://arxiv.org/abs/2502.11271.

Lù, X. H., Kazemnejad, A., Meade, N., Patel, A., Shin, D., Zambrano, A., Stańczak, K., Shaw, P., Pal, C. J., and Reddy, S. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories. *arXiv preprint arXiv:2504.08942*, 2025. URL https://arxiv.org/abs/2504.08942.

Ma, S., Gao, B., Jia, X., Qin, S., Li, T., Ma, K., Jia, X., Ren, W., and Liu, Y. Odar: Principled adaptive routing for llm reasoning via active inference. *arXiv preprint arXiv:2602.23681*, 2026. URL https://arxiv.org/abs/2602.23681.

Ma, W., Nan, J., Wu, W., and Chen, Y. What deserves memory: Adaptive memory distillation for llm agents. *arXiv preprint arXiv:2508.03341*, 2025. URL https://arxiv.org/abs/2508.03341.

Maiorano, A. C. Llm readiness harness: Evaluation, observability, and ci gates for llm/rag applications. *arXiv preprint arXiv:2603.27355*, 2026. URL https://arxiv.org/abs/2603.27355.

Merrill, M. A. et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in terminal environments. *arXiv preprint arXiv:2601.11868*, 2026. URL https://arxiv.org/abs/2601.11868.

Mialon, G., Fourrier, C., Swift, C., Wolf, T., LeCun, Y., and Scialom, T. Gaia: A benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*, 2023. URL https://arxiv.org/abs/2311.12983.

Moslemi, Z., Koneru, K., Lee, Y.-T., Kumar, S., and Radhakrishnan, R. Polaris: Typed planning and governed execution for agentic ai in back-office automation. *arXiv preprint arXiv:2601.11816*, 2026. URL https://arxiv.org/abs/2601.11816.

Natarajan, S., Mathur, S., Sidheekh, S., Stammer, W., and Kersting, K. Human-in-the-loop or ai-in-the-loop? automate or collaborate? *arXiv preprint arXiv:2412.14232*, 2024. URL https://arxiv.org/abs/2412.14232.

Nathani, D., Madaan, L., Roberts, N., Bashlykov, N., Menon, A., Moens, V., Budhiraja, A., Magka, D., Vorotilov, V., Chaurasia, G., Hupkes, D., Cabral, R. S., Shavrina, T., Foerster, J., Bachrach, Y., Wang, W. Y., and Raileanu, R. Mlgym: A new framework and benchmark for advancing ai research agents. *arXiv preprint arXiv:2502.14499*, 2025. URL https://arxiv.org/abs/2502.14499.

Ndzomga, F. Efficient benchmarking of ai agents. *arXiv preprint arXiv:2603.23749*, 2026. URL https://arxiv.org/abs/2603.23749.

Ni, Z., Xiao, Y., and Liang, Z. Agentified benchmarking for logical reasoning agents. In *ICLR Workshop on Agents in the Wild*, 2026. URL https://openreview.net/forum?id=LAmfBVlm7b.

OpenAI. Learning to reason with LLMs, 2024. URL https://openai.com/index/learning-to-reason-with-llms/.

OpenAI. Reasoning models — OpenAI API guide, 2026. URL https://developers.openai.com/api/docs/guides/reasoning/.

Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., and Gonzalez, J. E. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

Park, Y. B. Graph-native cognitive memory for ai agents: Formal belief revision semantics for versioned memory architectures. *arXiv preprint arXiv:2603.17244*, 2026. URL https://arxiv.org/abs/2603.17244.

Phan, L. et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025. URL https://arxiv.org/abs/2501.14249.

Qin, Y., Ye, Y., Fang, J., Wang, H., Liang, S., Tian, S., Zhang, J., Li, J., Li, Y., Huang, S., et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025. URL https://arxiv.org/abs/2501.12326.

Rabanser, S., Kapoor, S., Kirgis, P., Liu, K., Utpala, S., and Narayanan, A. Towards a science of ai agent reliability. *arXiv preprint arXiv:2602.16666*, 2026. URL https://arxiv.org/abs/2602.16666.

Rasmussen, P., Paliychuk, P., Beauvais, T., Ryan, J., and Chalef, D. Zep: A temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025. URL https://arxiv.org/abs/2501.13956.

Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., and Bowman, S. R. GPQA: A graduate-level google-proof Q&A benchmark. In *COLM*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Roman, A. and Roman, J. Orchestral ai: A framework for agent orchestration. *arXiv preprint arXiv:2601.02577*, 2026. URL https://arxiv.org/abs/2601.02577.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, 2023. URL https://openreview.net/forum?id=Yacmpz84TH.

Shi, Q., Zytek, A., Razavi, P., Narasimhan, K., and Barres, V. τ-Knowledge: Evaluating conversational agents over unstructured knowledge. *arXiv preprint arXiv:2603.04370*, 2026. URL https://arxiv.org/abs/2603.04370.

Siegel, Z. S., Kapoor, S., Nagdir, N., Stroebl, B., and Narayanan, A. Core-bench: Fostering the credibility of published research through a computational reproducibility agent benchmark. *arXiv preprint arXiv:2409.11363*, 2024. URL https://arxiv.org/abs/2409.11363.

Sun, W., Lu, M., Ling, Z., Liu, K., Yao, X., Yang, Y., and Chen, J. Scaling long-horizon llm agent via context-folding. *arXiv preprint arXiv:2510.11967*, 2025. URL https://arxiv.org/abs/2510.11967.

Suzgun, M., Scales, N., Schärli, N., Gehrmann, S., Wei, J., Chiang, H. W., Bosma, M., Zhou, D., Le, Q. V., Chi, E. H., et al. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022. URL https://arxiv.org/abs/2210.09261.

Suzgun, M., Yuksekgonul, M., Bianchi, F., Jurafsky, D., and Zou, J. Dynamic cheatsheet: Test-time learning with adaptive memory. In *EACL*, 2026. URL https://aclanthology.org/2026.eacl-long.333/.

Vasilopoulos, A. Codified context: Infrastructure for ai agents in a complex codebase. *arXiv preprint arXiv:2602.20478*, 2026. URL https://arxiv.org/abs/2602.20478.

Verma, N. Active context compression: Autonomous memory management in llm agents. *arXiv preprint arXiv:2601.07190*, 2026. URL https://arxiv.org/abs/2601.07190.

Vuddanti, S. V. and Chittiprolu, S. K. Recoverability has a law: The err measure for tool-augmented agents. *arXiv preprint arXiv:2601.22352*, 2026. URL https://arxiv.org/abs/2601.22352.

Wan, G., Ling, M., Ren, X., Han, R., Li, S., and Zhang, Z. Compass: Enhancing agent long-horizon reasoning with evolving context. *arXiv preprint arXiv:2510.08790*, 2025. URL https://arxiv.org/abs/2510.08790.

Wang, X., Wei, J., Schuurmans, D., Le, Q. V., Chi, E. H., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. In *ICLR*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Wang, X., Hu, Z., Lu, P., Zhu, Y., Zhang, J., Subramaniam, S., Loomba, A. R., Zhang, S., Sun, Y., and Wang, W. Scibench: Evaluating college-level scientific problem-solving abilities of large language models. In *ICML*, 2024. URL https://proceedings.mlr.press/v235/wang24z.html.

Wang, Y., Shi, Y., Yang, M., Zhang, R., He, S., Lian, H., Chen, Y., Ye, S., Cai, K., and Gu, X. Swe-pruner: Self-adaptive context pruning for coding agents. *arXiv preprint arXiv:2601.16746*, 2026. URL https://arxiv.org/abs/2601.16746.

Wang, Z., Chang, Q., Patel, H., Biju, S., Wu, C.-E., Liu, Q., Ding, A., Rezazadeh, A., Shah, A., Bao, Y., and Siow, E. Mcp-bench: Benchmarking tool-using LLM agents with complex real-world tasks via MCP servers. *arXiv preprint arXiv:2508.20453*, 2025a. URL https://arxiv.org/abs/2508.20453.

Wang, Z. Z., Gandhi, A., Neubig, G., and Fried, D. Inducing programmatic skills for agentic tasks. *arXiv preprint arXiv:2504.06821*, 2025b. URL https://arxiv.org/abs/2504.06821.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichtier, B., Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. Chain of thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022. URL https://arxiv.org/pdf/2201.11903.

Wei, J., Sun, Z., Papay, S., McKinney, S., Han, J., Fulford, I., Chung, H. W., Passos, A. T., Fedus, W., and Glaese, A. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025a. URL https://arxiv.org/abs/2504.12516.

Wei, T., Sachdeva, N., Coleman, B., He, Z., Bei, Y., Ning, X., Ai, M., Li, Y., He, J., Chi, E. H., Wang, C., Chen, S., Pereira, F., Kang, W.-C., and Cheng, D. Z. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*, 2025b. URL https://arxiv.org/abs/2511.20857.

Wu, J., Yin, W., Jiang, Y., Wang, Z., Xi, Z., Fang, R., Zhang, L., He, Y., Zhou, D., Xie, P., and Huang, F. Webwalker: Benchmarking llms in web traversal. *arXiv preprint arXiv:2501.07572*, 2025. URL https://arxiv.org/abs/2501.07572.

Wu, Y., Fu, D., Si, W., Huang, Z., Jiang, M., Li, K., Xia, S., Sun, J., Xu, T., Xiao, Y., Lu, P., Cai, X., Ye, L., Zhu, W., Hu, X., and Liu, P. Innovatorbench: Evaluating agents' ability to conduct innovative ai research. In *ICLR*, 2026. URL https://openreview.net/forum?id=w8rZ2Jd6Jo.

Xiao, Y.-A., Gao, P., Peng, C., and Xiong, Y. Reducing cost of llm agents with trajectory reduction. *Proceedings of the ACM on Software Engineering*, 2026. URL https://arxiv.org/abs/2509.23586.

Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., Hua, T. J., Cheng, Z., Shin, D., Lei, F., Liu, Y., Xu, Y., Zhou, S., Savarese, S., Xiong, C., Zhong, V., and Yu, T. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024. URL https://arxiv.org/abs/2404.07972.

Xu, W., Liang, Z., Mei, K., Gao, H., Tan, J., and Zhang, Y. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025. URL https://arxiv.org/abs/2502.12110.

Yan, B. Y., Li, C., Qian, H., Lu, S., and Liu, Z. General agentic memory via deep research. *arXiv preprint arXiv:2511.18423*, 2025. URL https://arxiv.org/abs/2511.18423.

Yang, A., Team, Q., et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Yang, S., Han, S. C., Ma, X., Li, Y., Madani, M. R. G., and Hovy, E. Evotool: Self-evolving tool-use policy optimization in llm agents via blame-aware mutation and diversity-aware selection. *arXiv preprint arXiv:2603.04900*, 2026a. URL https://arxiv.org/abs/2603.04900.

Yang, Y., Li, J., Pan, Q., Zhan, B., Cai, Y., Du, L., Zhou, J., Chen, K., Chen, Q., Li, X., Zhang, B., and He, L. Autoskill: Experience-driven lifelong learning via skill self-evolution. *arXiv preprint arXiv:2603.01145*, 2026b. URL https://arxiv.org/abs/2603.01145.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. R. Tree of thoughts: Deliberate problem solving with large language models. In *NeurIPS*, 2023a. URL https://openreview.net/forum?id=5Xc1ecxO1h.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. React: Synergizing reasoning and acting in language models. In *ICLR*, 2023b. URL https://openreview.net/forum?id=WE_vluYUL-X.

Yao, S., Shinn, N., Razavi, P., and Narasimhan, K. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024. URL https://arxiv.org/abs/2406.12045.

Yoran, O., Amouyal, S. J., Malaviya, C., Bogin, B., Press, O., and Berant, J. Assistantbench: Can web agents solve realistic and time-consuming tasks? In *EMNLP*, 2024. URL https://arxiv.org/abs/2407.15711.

Yu, B., Zhu, Y., He, P., and Kang, D. Utboost: Rigorous evaluation of coding agents on swe-bench. In *ACL*, 2025a. URL https://aclanthology.org/2025.acl-long.189/.

Yu, Z., Peng, R., Ding, K., Li, Y., Peng, Z., Liu, M., Zhang, Y., Yuan, Z., Xin, H., Huang, W., Wen, Y., Zhang, G., and Liu, W. Formalmath: Benchmarking formal mathematical reasoning of large language models. *arXiv preprint arXiv:2505.02735*, 2025b. URL https://arxiv.org/abs/2505.02735.

Zahn, O. and Chana, S. Facts as first class objects: Knowledge objects for persistent llm memory. *arXiv preprint arXiv:2603.17781*, 2026. URL https://arxiv.org/abs/2603.17781.

Zhai, W. et al. Hle-verified: A systematic verification and structured revision of humanity's last exam. *arXiv preprint arXiv:2602.13964*, 2026. URL https://arxiv.org/abs/2602.13964.

Zhang, H., Long, Q., Bao, J., Feng, T., Zhang, W., Yue, H., and Wang, W. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026a. URL https://arxiv.org/abs/2602.02474.

Zhang, H., Yue, H., Feng, T., Long, Q., Bao, J., Jin, B., Zhang, W., Li, X., You, J., Qin, C., and Wang, W. Learning query-aware budget-tier routing for runtime agent memory. *arXiv preprint arXiv:2602.06025*, 2026b. URL https://arxiv.org/abs/2602.06025.

Zhang, L., Chen, S., Cai, Y., others, and E, W. Bohrium + scimaster: Building the infrastructure and ecosystem for agentic science at scale. *arXiv preprint arXiv:2512.20469*, 2025a. URL https://arxiv.org/abs/2512.20469.

Zhang, Q., Hu, C., Upasani, S., Ma, B., Hong, F., Kamanuru, V., Rainton, J., Wu, C., Ji, M., Li, H., Thakker, U., Zou, J., and Olukotun, K. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025b. URL https://arxiv.org/abs/2510.04618.

Zhang, R., Zhao, X., Wang, R., Chen, S., Zhang, G., Zhang, A., Wang, K., and Wen, Q. Safesieve: From heuristics to experience in progressive pruning for llm-based multi-agent communication. *arXiv preprint arXiv:2508.11733*, 2025c. URL https://arxiv.org/abs/2508.11733.

Zhang, S., Wang, J., Zhou, R., Liao, J., Feng, Y., Zhang, W., Wen, Y., Li, Z., Xiong, F., Qi, Y., Tang, B., and Wen, M. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*, 2026c. URL https://arxiv.org/abs/2601.03192.

Zheng, J., Shi, C., Cai, X., Li, Q., Zhang, D., Li, C., Yu, D., and Ma, Q. Lifelong learning of large language model based agents: A roadmap. *arXiv preprint arXiv:2501.07278*, 2025a. URL https://arxiv.org/abs/2501.07278.

Zheng, Y., Hu, Y., Yu, T., and Quinn, A. Agentsight: System-level observability for ai agents using ebpf. *arXiv preprint arXiv:2508.02736*, 2025b. URL https://arxiv.org/abs/2508.02736.

Zheng, Y., Li, C., Li, S., Zhang, Y., and Su, Z. Agentic harness for real-world compilers. *arXiv preprint arXiv:2603.20075*, 2026. URL https://arxiv.org/abs/2603.20075.

Zhou, A., Yan, K., Shlapentokh-Rothman, M., Wang, H., and Wang, Y.-X. Language agent tree search unifies reasoning acting and planning in language models. In *ICML*, 2024. URL https://proceedings.mlr.press/v235/zhou24r.html.

Zhou, C., Chai, H., Chen, W., Guo, Z., Shan, R., Song, Y., Xu, T., Yang, Y., Yu, A., Zhang, W., Zheng, C., Zhu, J., Zheng, Z., Zhang, Z., Lou, X., Zhang, C., Fu, Z., Wang, J., Liu, W., Lin, J., and Zhang, W. Externalization in llm agents: A unified review of memory, skills, protocols and harness engineering. *arXiv preprint arXiv:2604.08224*, 2026a. URL https://arxiv.org/abs/2604.08224.

Zhou, H., Chen, Y., Guo, S., Yan, X., Lee, K. H., Wang, Z., Lee, K. Y., Zhang, G., Shao, K., Yang, L., and Wang, J. Memento: Fine-tuning llm agents without fine-tuning llms. *arXiv preprint arXiv:2508.16153*, 2025a. URL https://arxiv.org/abs/2508.16153.

Zhou, Q., Zhang, J., Wang, H., Hao, R., Wang, J., Han, M., Yang, Y., Wu, S., Pan, F., Fan, L., Tu, D., and Zhang, Z. Featurebench: Benchmarking agentic coding for complex feature development. In *ICLR*, 2026b. URL https://openreview.net/forum?id=41xrZ3uGuI.

Zhou, S., Xu, F. F., Zhu, H., Yuan, X., Gong, C., Lai, C., Chen, Z., Zhu, T., Wang, R., Lei, L., et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023. URL https://arxiv.org/abs/2307.13854.

Zhou, Z., Cao, C., Feng, X., Li, X., Li, Z., Lu, X., Yao, J., Huang, W., Cheng, T., Zhang, J., Jiang, T., Xu, L., Zheng, Y., Miranda, B., Liu, T., Koyejo, S., Sugiyama, M., and Han, B. AlphaApollo: A system for deep agentic reasoning. *arXiv preprint arXiv:2510.06261*, 2025b. URL https://arxiv.org/abs/2510.06261.

Zhu, K., Du, H., Hong, Z., Yang, X., Guo, S., Wang, Z., Wang, Z., Qian, C., Tang, X., Ji, H., and You, J. Multiagentbench: Evaluating the collaboration and competition of llm agents. In *ACL*, 2025a. URL https://aclanthology.org/2025.acl-long.421/.

Zhu, K., Liu, Z., Li, B., Tian, M., Yang, Y., Zhang, J., Han, P., Xie, Q., Cui, F., Zhang, W., Ma, X., Yu, X., Ramesh, G., Wu, J., Liu, Z., Lu, P., Zou, J., and You, J. Where llm agents fail and how they can learn from failures. *arXiv preprint arXiv:2509.25370*, 2025b. URL https://arxiv.org/abs/2509.25370.

Zhu, Q., Chen, S., Yu, R., Wu, Z., and Wang, B. From lossy to verified: A provenance-aware tiered memory for agents. *arXiv preprint arXiv:2602.17913*, 2026. URL https://arxiv.org/abs/2602.17913.

A. Further Discussions

Broader impacts. AlphaDiana aims to improve the rigor, reproducibility, and interpretability of agent evaluation by making the model-harness boundary explicit and preserving auditable execution traces. This can help researchers and practitioners avoid misleading leaderboard-style comparisons, diagnose harness-induced failures, and make more informed claims about the capabilities and limitations of tool-using agents. These benefits are especially relevant for high-stakes uses of agents in software engineering, education, and scientific reasoning, where evaluation artifacts can shape deployment decisions. At the same time, a stronger evaluation infrastructure may accelerate the development of more capable autonomous agents, including systems that could be misused for harmful automation, deception, or unsafe code execution. Trajectory logging may also raise privacy and data-governance concerns when evaluations involve sensitive prompts, files, or tool outputs. We therefore encourage responsible use with sandboxed execution, access control, privacy-preserving logging, and transparent reporting of evaluation scope, assumptions, and limitations.

Limitations. AlphaDiana focuses on harness-aware evaluation for verifiable reasoning tasks, where deterministic or programmatic scorers provide reliable correctness signals. This scope improves reproducibility but does not cover many open-ended agent settings, such as long-form research, dialogue, negotiation, or subjective decision-making, where evaluation depends on human or model-based judgment. Our experiments also cover a small set of open models, harnesses, benchmarks, and ablation axes; broader conclusions may change with different agent frameworks, tool ecosystems, budgets, or deployment environments. In addition, while trajectory logging enables process-level analysis, our action taxonomy and failure attribution remain approximate and may miss finer-grained mechanisms. Finally, the current micro-ablations isolate externally visible capabilities such as tool exposure and skill loading but do not fully disentangle deeper harness components such as planning policies, memory management, retry logic, and answer extraction. Future work should extend AlphaDiana to richer agent tasks, more diverse harnesses, and more causal interventions over internal runtime design choices.

B. Related Work

This work studies *system-level* evaluation of agentic reasoning. Our starting point is that an open agent is best viewed not as a model alone, but as a *model-harness system*: the *model* provides raw generative and reasoning capacity, while the *harness* determines how that capacity is realized through control flow, tool access, context management, memory, interaction protocols, runtime policies, and feedback handling. This perspective connects several research threads: reasoning-capable foundation models, agent runtimes and tool-use frameworks, memory and context-management mechanisms, benchmark families ranging from model-centric reasoning tasks to interactive agent environments, and evaluation protocols that emphasize trajectories, robustness, and efficiency. It also clarifies a key distinction. *Model reasoning* studies a standalone model under direct inference, whereas *agentic reasoning* studies a model embedded in a multi-step loop that can act, observe, revise, and continue across turns. Once reasoning is coupled with action, environment interaction, and persistent state, the object of evaluation is no longer just the model, but the full system that turns the model into an agent. We therefore organize related work around the transition from *model-only* evaluation to *model-harness* evaluation.

B.1. Reasoning foundation models

What this line of work evaluates. A large body of work evaluates the reasoning ability of language models under direct or lightly structured inference. Chain-of-thought prompting showed that eliciting intermediate reasoning steps can substantially improve complex reasoning in large models (Wei et al., 2022), while benchmarks such as MATH and GSM8K became standard tests of mathematical reasoning (Hendrycks et al., 2021; Cobbe et al., 2021). Broader evaluation efforts such as HELM emphasized the importance of standardized, multi-metric, and scenario-diverse assessment of language models (Liang et al., 2022), while challenging reasoning-focused suites such as BIG-Bench Hard highlighted persistent gaps in compositional and symbolic reasoning even for strong language models (Suzgun et al., 2022). Subsequent work explored inference-time search and aggregation strategies, including self-consistency and tree-based reasoning, showing that measured reasoning quality depends not only on the base model but also on the inference procedure used to query it (Wang et al., 2023; Yao et al., 2023a). Open-weight reasoning models such as Qwen3 and DeepSeek-R1 further demonstrate that targeted post-training and reinforcement learning can substantially improve reasoning performance beyond standard instruction tuning (Yang et al., 2025; Guo et al., 2025a). In parallel, API-based reasoning models expose explicit deliberation or "thinking" modes and have become common baselines for complex reasoning and agentic workflows (OpenAI, 2024; 2026; Google, 2024).

Why this is still insufficient for harness-aware evaluation. These works are primarily evaluated at the *model level*, usually under direct prompting, fixed sampling budgets, and standardized inputs. That setting is appropriate when the target of study is the standalone model. However, in realistic use, strong reasoning increasingly depends on how a model is embedded in a runtime with tools, memory, observations, retries, and execution policies. As a result, this literature tells us a great deal about *raw model reasoning*, but much less about how reasoning changes once the model is wrapped by a harness. This creates a gap between *model reasoning* as it is usually benchmarked and *agentic reasoning* as it is actually deployed.

B.2. Agentic reasoning systems and open agents

What this line of work evaluates. Modern reasoning systems increasingly combine a foundation model with planning, tool use, memory, and environment interaction, producing long-horizon trajectories rather than single completions. Foundational examples include ReAct (Yao et al., 2023b), which interleaves reasoning and acting, LATS (Zhou et al., 2024), which performs search over action trajectories, and Toolformer, which trains models to invoke tools in a self-supervised manner (Schick et al., 2023). More broadly, recent work on externalized reasoning emphasizes that capabilities often emerge from interaction between a model and external resources rather than from the parametric model alone (Zhou et al., 2026a).

Open agent systems vary widely in how they realize this interaction. **Gateway- and runtime-centered** systems treat sessions, routing, and persistent state as first-class concerns; for example, OpenClaw embeds a model in a sessionful gateway with tool interfaces, memory, and policy controls. **Tool-oriented** frameworks emphasize modular tool abstractions, planning, and execution, aiming to improve reasoning through better external affordances rather than extra model training (Lu et al., 2025b). **Domain- and workflow-oriented** systems, especially in scientific settings, prioritize execution grounding and traceability (Zhang et al., 2025a). AlphaApollo similarly orchestrates multiple models and professional tools through explicit iteration and structured tool use (Zhou et al., 2025b). Recent surveys synthesize this broader shift from model-centric reasoning toward autonomous, lifelong, and self-improving agent systems (Zheng et al., 2025a; Ferrag et al., 2025).

Why this is still insufficient for harness-aware evaluation. This literature establishes that many agent capabilities arise from the surrounding runtime rather than from the base model alone. But it typically studies complete systems, methods, or design patterns, rather than isolating the harness as an explicit object of evaluation. As a result, two agents built on the same model may behave very differently because of differences in orchestration, tool routing, state management, or feedback handling, yet those differences are often reported only as properties of the overall system. For our purposes, this means the key missing step is not another agent architecture, but a framework that can compare *matched model-harness systems* and attribute performance differences to the harness itself.

Memory and context management. A central source of variation across agent systems is how they manage context windows and persistent state. Early work on long-context efficiency focused on compression and memory virtualization, for example LongLLMLingua for prompt compression (Jiang et al., 2024) and MemGPT for OS-inspired hierarchical memory management (Packer et al., 2023). Subsequent work introduced retrieval- and structure-based long-term memory, including HippoRAG (Gutierrez et al., 2024), temporal knowledge-graph memory in Zep (Rasmussen et al., 2025), object-centric persistent memory (Zahn & Chana, 2026), graph-native versioned memory architectures (Park, 2026), provenance-aware verified memory (Zhu et al., 2026), and governance-oriented memory control (Kumar et al., 2026). A second line of work studies working-memory management for long-horizon agents, including hierarchical working memory in HiAgent (Hu et al., 2025), agentic memory modules such as A-MEM and Mem0 (Xu et al., 2025; Chhikara et al., 2025), adaptive memory distillation and memory-structure selection (Ma et al., 2025; Lu et al., 2026), query-aware budget-tier routing (Zhang et al., 2026b), and compression or pruning methods such as ACON, Active Context Compression, SWE-Pruner, and trajectory reduction (Kang et al., 2025; Verma, 2026; Wang et al., 2026; Xiao et al., 2026). Other work emphasizes evolving contexts and summarization-based management, including COMPASS, context folding, and end-to-end summarization for multi-turn RL (Wan et al., 2025; Sun et al., 2025; Lu et al., 2025a), as well as efficient lifelong and self-evolving memory systems such as Dynamic Cheatsheet, SimpleMem, MemRL, MemSkill, and General Agentic Memory (Suzgun et al., 2026; Liu et al., 2026b; Zhang et al., 2026c;a; Yan et al., 2025).

What this line of work evaluates is how different memory mechanisms improve long-horizon reasoning, retrieval, persistence, or adaptation. *Why this is still insufficient* is that memory is usually treated as a design improvement inside a particular agent, rather than as a controlled harness factor whose contribution can be measured under matched models, tasks, and budgets. For our framing, memory is not merely a model property; it is a harness choice that can substantially alter performance even when the base model is fixed.

Routing, orchestration, planning, and budgeted control. Another major aspect of the harness concerns how computation is allocated and how decisions are organized across steps, tools, and sub-agents. Planning-oriented approaches such as Plan-and-Act improve long-horizon behavior through explicit planning before execution (Erdogan et al., 2025). Recent work extends this into broader orchestration and adaptive control, including reflective prompt evolution in GEPA (Agrawal et al., 2025), generalized routing across models and agents (Guo et al., 2025b), evolving contexts for self-improving agents (Zhang et al., 2025b), orchestration frameworks such as Orchestral AI (Roman & Roman, 2026), context-aware multi-agent routing in CASTER (Liu et al., 2026c), intention-based planning under costly tool budgets (Liu et al., 2026a), and adaptive routing via active inference in ODAR (Ma et al., 2026).

What this line of work evaluates is how better control policies improve long-horizon behavior, budget allocation, or tool-use efficiency. *Why this is still insufficient* is that these methods usually optimize a specific system rather than support controlled comparison across different harnesses under a shared protocol. Together, these works show that the harness is not a passive wrapper but an active policy layer that determines planning depth, routing decisions, budget allocation, and fallback behavior. What remains underexplored is how much these factors contribute to measured reasoning performance when the underlying model is held fixed.

Skills, tool abstraction, and compositional reasoning. A related line of work studies *skills* as higher-level abstractions that sit between a model and raw tool calls. Examples include induced programmatic skills for agentic tasks (Wang et al., 2025b), compositional skill synthesis for reasoning (Jiao et al., 2026), empirical analyses of deployed skill ecosystems (Ling et al., 2026), lifelong skill self-evolution (Yang et al., 2026b), and token-efficient skill optimization (Gao et al., 2026).

What this line of work evaluates is how structured action abstractions can make agent behavior more effective or efficient. *Why this is still insufficient* is that skills reshape the effective action space and control flow seen by the model, yet are rarely analyzed as part of a broader harness decomposition. These abstractions therefore naturally belong to the harness layer rather than to the base model alone, and should be evaluated as such.

Self-improving and self-evolving agents. The same is true for systems that adapt at test time through experience, memory updates, tool-policy updates, or self-modification. Recent work studies EvoAgent (Feng et al., 2025), self-evolving multi-agent simulation frameworks (Almansoori et al., 2025), SafeSieve for experience-driven pruning in multi-agent communication (Zhang et al., 2025c), EvoTool for self-evolving tool-use policies (Yang et al., 2026a), and Memento, which improves agents by tuning the surrounding system rather than the base model itself (Zhou et al., 2025a). Test-time learning is also explored through adaptive memory and human guidance, as in Dynamic Cheatsheet and human-in-the-loop self-improvement methods (Suzgun et al., 2026; He et al., 2025), while Evo-Memory makes self-evolving memory an explicit evaluation target (Wei et al., 2025b).

What this line of work evaluates is how agents can improve during deployment through adaptation of memory, policy, or surrounding scaffolds. *Why this is still insufficient* is that these gains are often reported as improvements in the overall agent, without disentangling whether the change comes from better reasoning, better adaptation mechanisms, or more favorable harness policies. These works reinforce the same point: gains in agent performance often arise through harness adaptation rather than parametric model changes alone.

Harness engineering, runtime infrastructure, and observability. A recent and rapidly growing body of work studies the harness layer itself as a systems problem. Some work focuses on observability, proposing system-level tracing and structured logs for agent execution, such as AgentSight and AgentTrace (Zheng et al., 2025b; AlSayyad et al., 2026). Other work studies governed or typed execution, hierarchical retrieval interfaces, or infrastructure for complex codebase interaction, for example POLARIS, A-RAG, and Codified Context (Moslemi et al., 2026; Du et al., 2026; Vasilopoulos, 2026). There is also growing interest in automatically constructing or optimizing the harness, including AutoHarness, domain-specific harnesses for terminal coding and compilers, and explicit optimization of model-harness combinations in Meta-Harness (Lou et al., 2026; Bui, 2026; Zheng et al., 2026; Lee et al., 2026). Recent conceptual work makes this systems view explicit by treating harness engineering as a distinct layer of control, runtime, and agency for language agents (He et al., 2026).

What this line of work evaluates is the harness as a systems artifact: something that can be instrumented, governed, synthesized, and optimized. *Why this is still insufficient* is that most of this work still does not provide a general evaluation substrate for matched comparison of model-harness systems on tasks with shared verifiers and budgets. If harnesses can be traced, governed, synthesized, and optimized as modular artifacts, then they should also be evaluated as such.

Security and safety. Because agent behavior is mediated by execution environments, tools, memory, and runtime policies, security and safety arise at the harness level rather than purely at the model level. Recent work on OpenClaw emphasizes security controls in session-based tool execution, while broader studies analyze how to secure agent execution under realistic runtime threats and failure modes (Buehler et al., 2025).

What this line of work evaluates is whether runtime policies, containment, and execution safeguards can make agents safer in practice. *Why this is still insufficient* is that safety-oriented harness choices can also affect measured task performance, making it difficult to interpret end-to-end scores unless policy enforcement and reasoning ability are considered together. This perspective reinforces the need for evaluation settings that can separate raw reasoning ability from policy enforcement, containment, and execution safeguards.

B.3. Benchmarks from model-centric reasoning to interactive agent evaluation

Benchmarking work relevant to our setting falls into two broad categories: **model-centric reasoning benchmarks** and **interactive agent benchmarks**. This distinction mirrors the transition studied in our paper: from evaluating a model under direct inference to evaluating a model embedded in a harness that can act, observe, and interact with an environment.

Model-centric reasoning benchmarks. A large body of work evaluates language models on mathematical, scientific, code, and formal reasoning under a prompt-in/answer-out protocol. Representative examples include MATH (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), GPQA (Rein et al., 2024), SciBench (Wang et al., 2024), BBH (Suzgun et al., 2022), HumanEval (Chen et al., 2021), and LiveCodeBench (Jain et al., 2024). Broader model-evaluation efforts such as HELM also highlight the importance of standardized evaluation across scenarios and metrics (Liang et al., 2022). More recent frontier and verifier-backed benchmarks extend this paradigm while retaining relatively clean correctness criteria. Humanity's Last Exam is designed to remain difficult at the knowledge frontier while preserving closed-ended grading (Phan et al., 2025), and HLE-Verified was introduced to improve reliability and reduce ambiguity through structured verification (Zhai et al., 2026). FrontierMath similarly emphasizes advanced mathematical reasoning with automated verification to reduce ambiguity and contamination (Glazer et al., 2024), while FormalMATH targets formal mathematical reasoning in theorem-proving-style settings (Yu et al., 2025b).

What this line of work evaluates is raw reasoning capability under controlled inference and relatively reliable judging. *Why this is still insufficient* is that most model-centric reasoning benchmarks assume a fixed inference protocol and evaluate the model as the primary object of interest. They are therefore ideal for measuring *raw model reasoning*, but do not by themselves distinguish model capability from harness capability once reasoning is mediated by tools, memory, execution support, and multi-step interaction.

Interactive agent benchmarks. A second line of work evaluates agents through action in realistic environments rather than through answer matching alone. Early and influential examples include AgentBench (Liu et al., 2024), Mind2Web (Deng et al., 2023), WebArena (Zhou et al., 2023), VisualWebArena (Koh et al., 2024), and WebVoyager (He et al., 2024), which study web interaction, browsing, and multimodal web-based tasks. Other benchmarks target broader forms of digital and workplace interaction, including OSWorld for open computer-use tasks (Xie et al., 2024), WorkArena for common knowledge-work tasks (Drouin et al., 2024), UI-TARS for native GUI interaction (Qin et al., 2025), and GAIA for general AI assistants solving realistic multi-step tasks (Mialon et al., 2023). A large coding- and software-oriented line includes SWE-bench for resolving real GitHub issues (Jimenez et al., 2024), Terminal-Bench for shell-based workflows (Merrill et al., 2026), ToolSandbox for stateful conversational tool use (Lu et al., 2024), R2E-Gym for procedural software-engineering environments with hybrid verifiers (Jain et al., 2025), ProdCodeBench for production-derived coding-agent evaluation (Jha et al., 2026), and FeatureBench for complex feature-level agentic coding tasks (Zhou et al., 2026b). Additional benchmarks emphasize realistic assistant, research, or collaborative-agent settings, including AssistantBench (Yoran et al., 2024), BrowseComp (Wei et al., 2025a), CORE-Bench for computational reproducibility (Siegel et al., 2024), DeepResearchGym (Coelho et al., 2025), ResearchGym (Garikaparthi et al., 2026), MLGym (Nathani et al., 2025), InnovatorBench for AI research agents (Wu et al., 2026), MultiAgentBench for collaboration and competition among multiple agents (Zhu et al., 2025a), and Agentified Benchmarking for Logical Reasoning Agents (Ni et al., 2026). Newer multi-turn and workflow-oriented settings such as τ-bench, WebWalker, NESTFUL, and τ-Knowledge place even greater emphasis on structured tool use, conversational state changes, and policy-constrained interaction (Yao et al., 2024; Wu et al., 2025; Basu et al., 2024; Shi et al., 2026). Recent work also argues that sufficiently rich computer environments can elicit broader forms of agentic intelligence than static prompting alone (Cheng et al., 2026).

What this line of work evaluates is end-to-end behavior of agents acting in realistic environments. *Why this is still insufficient* is that end-to-end performance depends not only on the base model, but also on the surrounding harness, including tool interfaces, control flow, observation formatting, memory and state handling, and runtime policies. This makes interactive agent benchmarks highly valuable for measuring full-system behavior, while also motivating harness-aware evaluation frameworks that can better separate model effects from harness effects.

Taken together, these two benchmark families motivate the core distinction in our paper. *Model-centric reasoning benchmarks* evaluate a model in isolation under direct inference, whereas *interactive agent benchmarks* evaluate a model-harness system acting in an environment. Our empirical design focuses on largely verifiable reasoning settings because they provide a clean testbed for connecting these two traditions: they retain reliable correctness signals from model-centric evaluation while surfacing the system-level effects that arise once a model is embedded in a harness.

B.4. Evaluation protocols, reliability, and human-agent interaction

What this line of work evaluates. A growing body of work argues that agent evaluation cannot be reduced to final accuracy alone. Agent behavior is shaped by planning quality, tool choice, policy compliance, recoverability, robustness, efficiency, and interaction quality. AgentBench evaluates LLMs as agents across diverse environments and highlights gaps in long-horizon reasoning and instruction following (Liu et al., 2024). T-Eval evaluates tool use *step by step* rather than only through final outcomes (Chen et al., 2024). τ-bench proposes reliability-oriented metrics over repeated trials in dynamic tool-agent-user interactions (Yao et al., 2024). MCP-Bench emphasizes trajectory-level planning and cross-tool coordination through standardized tool servers (Wang et al., 2025a). OSWorld-Human complements success metrics with human reference trajectories and efficiency diagnostics (Abhyankar et al., 2025). AgentRewardBench further studies the reliability of automatic trajectory evaluation itself, focusing on how web-agent trajectories should be judged rather than only how agents should act (Lù et al., 2025).

Recent work expands this in several directions. Some papers focus on debugging and recoverability, including failure analysis for agent trajectories and explicit metrics such as ERR (Zhu et al., 2025b; Vuddanti & Chittiprolu, 2026). Others study robustness under more realistic interfaces, such as imperfect APIs, context retrieval in coding agents, and memory stress tests in long-horizon interactions (Kim et al., 2026; Li et al., 2026; Hu et al., 2026; Jiayang et al., 2026). A broader reliability perspective is developed in recent work calling for a more systematic science of AI agent reliability, alongside practical regression-testing and readiness frameworks such as AgentAssay and LLM Readiness Harness (Rabanser et al., 2026; Bhardwaj, 2026; Maiorano, 2026). Related work also highlights evaluation variance and protocol sensitivity, for example through analyses of randomness in agentic evaluation (Bjarnason et al., 2026) and through more rigorous evaluation protocols for agents on SWE-bench (Yu et al., 2025a). Related work also studies when human collaboration is preferable to full automation and how human feedback can improve agent behavior at test time (Natarajan et al., 2024; He et al., 2025).

Why this is still insufficient for harness-aware evaluation. Collectively, these protocols motivate trajectory-aware logging, reproducibility mechanisms, modular judging, and reliability analysis beyond a single scalar success metric. However, most of this work aims to improve how end-to-end agents are evaluated, rather than to isolate what portion of measured performance should be attributed to the harness itself. In other words, it strengthens the measurement of agent behavior, but does not by itself resolve the attribution problem created when the evaluated object is a model embedded in a complex runtime.

B.5. Open gap: from model-only to model-harness evaluation

What remains missing. Despite substantial progress on reasoning models, memory systems, routing and orchestration, skill abstractions, benchmark suites, reliability analyses, and runtime infrastructure, the field still lacks a unified system-level evaluation substrate that composes heterogeneous agents, benchmark drivers and judges, and execution backends under controlled policies while producing replayable trajectories. In practice, evaluation code is often tied to a specific harness, runtime, or tool protocol, making cross-system comparison brittle and difficult to reproduce. This creates a persistent attribution problem: the same model may appear stronger or weaker depending on the surrounding scaffold, while end-to-end benchmark scores often conflate model capability with orchestration quality, context management, or execution policy.

How our work positions itself. Our work targets this gap directly. We seek to move from *model-only* evaluation to *model-harness* evaluation by treating the agent as the object of study at the macro level while also analyzing the harness as a decomposable source of performance differences at the micro level. In particular, we focus on open agents evaluated in

controlled, largely verifiable reasoning settings, where correctness can be checked deterministically and harness effects can be studied more cleanly. AlphaDiana is designed as a unified substrate for this purpose: it provides common abstractions for agent adapters, benchmark drivers, sandbox backends, isolated judges, and trajectory-level observability, enabling controlled comparisons such as "Qwen3-8B + OpenClaw on HLE" without entangling benchmark logic with agent integration. This goal is closely aligned with broader calls for modular, transparent, and reproducible evaluation infrastructure in both model-centric and agentic settings (Liang et al., 2022; Kapoor et al., 2026). At the same time, recent benchmarks for AI research, logical reasoning agents, coding agents, and reproducibility-oriented agents further illustrate that the community is rapidly expanding what counts as an agent evaluation target (Siegel et al., 2024; Garikaparthi et al., 2026; Wu et al., 2026; Jha et al., 2026; Zhou et al., 2026b; Ni et al., 2026). What remains underdeveloped is a framework for *matched* comparison across open model-harness systems, where the model, harness, benchmark, verifier, and budget policy can be varied in controlled ways. In this sense, AlphaDiana is not merely another benchmark harness, but an infrastructure layer for systematic *model-harness* evaluation.

C. AlphaDiana

We present **AlphaDiana**, a modular framework for *system-level evaluation* of LLM-based reasoning systems. AlphaDiana is designed for settings in which performance depends not only on the base models, but also on the surrounding runtime stack, including the agent harnesses, tool interfaces, execution environments, scoring protocols, and run configurations. The framework factorizes an evaluation into independent modules with explicit interfaces, enabling controlled comparison across direct models and harnessed-agent systems.

C.1. Overview

Evaluation tuple. An experiment executed by AlphaDiana is formalized as

$$\mathcal{R} = \mathcal{U}(\mathcal{B}, f_\theta, \mathcal{H}, \mathcal{E}, \mathcal{S}, \mathcal{C}),$$

where $\mathcal{R}$ is the result records of the experiment, $\mathcal{U}$ is the runner that orchestrates the experiment, $\mathcal{B}$ is the benchmark, f_θ is the base model, $\mathcal{H}$ is the harness, $\mathcal{E}$ is an optional execution environment, $\mathcal{S}$ is the scorer, and $\mathcal{C}$ is the run configuration. The agent system is denoted by

$$\mathcal{A}_{\theta,\mathcal{H}} = \mathcal{H} \odot f_\theta,$$

where $\odot$ indicates that the harness wraps the base model by providing prompts, tools, state management, control logic, and answer extraction. The direct-model baseline is the special case in which $\mathcal{H}$ reduces to minimal task formatting without an external tool loop.

Modular decomposition. AlphaDiana separates evaluation into four user-specified components (benchmark, agent, environment, scorer) and one orchestration layer (runner). Specifically, the benchmark poses tasks, the agent solves them, the environment executes tool-call requests when tool use is required, the scorer judges the resulting prediction, and the runner coordinates the full protocol. The runner executes the pipeline

$$\texttt{load task} \longrightarrow \texttt{solve} \longrightarrow \texttt{score} \longrightarrow \texttt{save results}.$$

This separation is central to the framework: benchmark-specific logic is not embedded in the harness, scorer-specific normalization is not delegated to the agent, and environment-specific execution details are isolated behind a sandbox interface. As a result, the same benchmark and scorer can be reused across multiple model-harness systems, making observed differences more attributable to the evaluated system rather than to hidden evaluation-side implementation choices.

Interaction loop. For a task instance $(x, y) \in \mathcal{B}$, the harness constructs the initial model context

$$p_1 = \mathrm{Prompt}_{\mathcal{H}}(x),$$

and initializes any runtime state

$$m_1 = \mathrm{InitState}_{\mathcal{H},\mathcal{E}}(x, \mathcal{C}).$$

At turn t, the model produces

$$o_t = f_\theta(p_t).$$

If o_t contains a final answer, the harness extracts the model-predicted answer $\hat{y}$ and terminates.

Notation	Meaning
$\mathcal{B}$	Benchmark, represented as a finite collection of task instances.
(x, y)	Task input x and reference object y.
f_θ	Base language model with parameters θ.
$\mathcal{H}$	Harness that prompts, controls, and operationalizes f_θ.
$\mathcal{A}_{\theta,\mathcal{H}}$	Agent system obtained by pairing f_θ with $\mathcal{H}$.
$\mathcal{E}$	Optional environment backend or sandbox for tool execution.
$\mathcal{S}$	Scorer that evaluates the agent's prediction.
$\mathcal{C}$	Run configuration specifying budgets, policies, and logging options.
$\mathcal{U}$	Runner that orchestrates the experiment of evaluation.
$\mathcal{R}$	Result records of the experiment.
p_t, o_t, a_t, r_t, m_t	Prompt, model output, parsed action, environment feedback, and runtime state at turn t.
$\hat{y}$	Final predicted answer.
τ	Complete interaction trajectory.
s	Task-level score or verdict.

Table 4. Core notation used in the AlphaDiana evaluation pipeline.

Otherwise, the harness parses an action

$$a_t = \mathrm{ParseAction}_{\mathcal{H}}(o_t),$$

dispatches it to the environment when admissible, and receives feedback

$$r_t = \mathcal{E}(a_t).$$

The harness then updates the runtime state and next prompt (context):

$$m_{t+1} = \mathrm{StateUpdate}_{\mathcal{H}}(m_t, o_t, r_t), \qquad p_{t+1} = \mathrm{PromptUpdate}_{\mathcal{H}}(p_t, o_t, r_t, m_{t+1}).$$

Thus, one harnessed interaction step can be summarized as

$$p_t \xrightarrow{f_\theta} o_t \xrightarrow{\mathcal{H}} a_t \xrightarrow{\mathcal{E}} r_t \xrightarrow{\mathrm{PromptUpdate}_{\mathcal{H}}} p_{t+1}.$$

Trajectory and scoring. The full trajectory is

$$\tau = \big((p_1, o_1, r_1), \ldots, (p_T, o_T, r_T), \hat{y}\big),$$

where T is the number of turns before termination. Termination occurs when the model emits a final answer, a budget in $\mathcal{C}$ is exhausted, or the harness or environment reaches a terminal error state. After termination, the scorer evaluates the prediction independently of the generation process:

$$s = \mathcal{S}(x, y, \hat{y}, \tau).$$

The runner then stores the prediction, score, trajectory, and metadata for later analysis.

Full procedure. Algorithm 1 summarizes the full evaluation procedure.

C.2. Benchmark Module

Role. The benchmark module loads a dataset and materializes it into standardized task instances. Formally, a benchmark is a finite collection

$$\mathcal{B} = \{(x_i, y_i)\}_{i=1}^{N},$$

where x_i is the task input and y_i is the reference object used for scoring. The reference object may be a string, numeric answer, symbolic expression, unit-test specification, verifier target, rubric, set of acceptable answers, or any other task-specific supervision object. AlphaDiana treats y_i abstractly: the benchmark provides the reference, while the scorer determines how it should be interpreted.

Task record. In implementation, each task record contains a unique task identifier, input x, reference field y, metadata, and optional attachments. Metadata may include difficulty, category, source, split, or expected output type. Attachments may include files, images, repositories, PDFs, or other artifacts required by multimodal or environment-based tasks. These fields may be utilized by the runner, agent, environment, or scorer, but they do not alter the core agent-facing interface.

Algorithm 1 The Evaluation Pipeline of AlphaDiana

Require: Benchmark $\mathcal{B}$, base model f_θ, harness $\mathcal{H}$, environment $\mathcal{E}$, scorer $\mathcal{S}$, configuration $\mathcal{C}$
Ensure: Result records $\mathcal{R}$ containing predictions, scores, trajectories, and metadata

Instantiate the agent system $\mathcal{A}_{\theta,\mathcal{H}} \leftarrow \mathcal{H} \odot f_\theta$
Initialize result store $\mathcal{R} \leftarrow \emptyset$
for each task instance $(x, y) \sim \mathcal{B}$ **do**
  Construct initial prompt $p_1 \leftarrow \mathrm{Prompt}_{\mathcal{H}}(x)$
  Initialize harness state $m_1 \leftarrow \mathrm{InitState}_{\mathcal{H},\mathcal{E}}(x, \mathcal{C})$
  Initialize trajectory $\tau \leftarrow \emptyset$, turn index $t \leftarrow 1$, final prediction $\hat{y} \leftarrow \varnothing$
  while not terminated under $\mathcal{C}$ **do**
    Generate model output $o_t \leftarrow f_\theta(p_t)$
    if o_t contains a final answer **then**
      Extract prediction $\hat{y} \leftarrow \mathrm{ExtractAnswer}_{\mathcal{H}}(o_t)$
      Append terminal record $(p_t, o_t, \varnothing)$ to τ
      break
    end if
    if o_t contains an admissible action **then**
      Parse action $a_t \leftarrow \mathrm{ParseAction}_{\mathcal{H}}(o_t)$
      Execute action and obtain feedback $r_t \leftarrow \mathcal{E}(a_t)$
    else
      Set feedback $r_t \leftarrow \mathrm{InvalidActionFeedback}_{\mathcal{H}}(o_t)$
    end if
    Append turn record (p_t, o_t, r_t) to τ
    Update harness state $m_{t+1} \leftarrow \mathrm{StateUpdate}_{\mathcal{H}}(m_t, o_t, r_t)$
    Update prompt/context $p_{t+1} \leftarrow \mathrm{PromptUpdate}_{\mathcal{H}}(p_t, o_t, r_t, m_{t+1})$
    Increment turn index $t \leftarrow t + 1$
  end while
  if $\hat{y} = \varnothing$ **then**
    Extract fallback prediction $\hat{y} \leftarrow \mathrm{FallbackExtract}_{\mathcal{H}}(\tau)$
  end if
  Score prediction $s \leftarrow \mathcal{S}(x, y, \hat{y}, \tau)$
  Collect metadata $z \leftarrow \mathrm{Metadata}(\mathcal{B}, f_\theta, \mathcal{H}, \mathcal{E}, \mathcal{S}, \mathcal{C}, \tau)$
  Save result $\mathcal{R} \leftarrow \mathcal{R} \cup \{(x, y, \hat{y}, s, \tau, z)\}$
end for
return $\mathcal{R}$

Module boundary. The benchmark specifies *what* problem is to be solved, not *how* it should be solved. It therefore does not contain harness-specific prompts, tool policies, memory rules, recovery strategies, or scoring logic. Prompt construction belongs to the harness, execution belongs to the environment, and correctness checking belongs to the scorer. This separation prevents benchmark definitions from implicitly favoring a particular harness design.

Interface. The abstract `Benchmark` interface exposes one required method:

`load_tasks(config)` $\rightarrow$ `list[BenchmarkTask]`.

This method converts a raw dataset, file collection, benchmark API, or generated task set into the standardized task sequence utilized by the runner. The configuration may specify split names, subset sizes, seeds, task filters, attachment paths, or benchmark-specific preprocessing options. Adding a new benchmark requires implementing this method and registering the benchmark under a configuration-visible name; no agent, environment, scorer, or runner code needs to change.

Listing 1. Benchmark interface (`base.py`, verbatim) and a minimal custom implementation. The `@register_benchmark` decorator registers the class under a string key that matches the `benchmark:` field in the YAML configuration.

```
# --- base.py (verbatim) ------------------------------------
@dataclass
class BenchmarkTask:
```

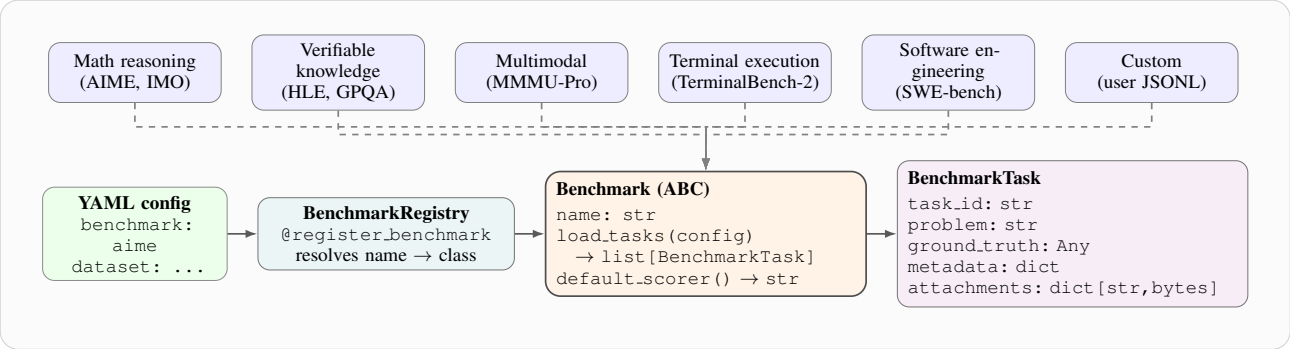


Figure 10. **Benchmark Module.** Concrete benchmark implementations (left) subclass the abstract `Benchmark` interface, which exposes `load_tasks()` to convert a raw dataset into a sequence of standardized `BenchmarkTask` tuples (right). The registry resolves a benchmark name from the YAML configuration to the corresponding implementation at runtime.

```
    task_id: str
    problem: str
    ground_truth: Any
    metadata: dict = field(default_factory=dict)
    attachments: dict[str, bytes] = field(default_factory=dict)

class Benchmark(ABC):
    name: str = ""

    @abstractmethod
    def load_tasks(self, config: dict) -> list[BenchmarkTask]: ...

    def default_scorer(self) -> str:
        return "exact_match"

# --- custom implementation (minimal example) --------------
@register_benchmark("my_benchmark")
class MyBenchmark(Benchmark):
    name = "my_benchmark"

    def load_tasks(self, config: dict) -> list[BenchmarkTask]:
        raw = load_my_dataset(config["dataset"])
        return [
            BenchmarkTask(
                task_id=item["id"],
                problem=item["question"],
                ground_truth=item["answer"],
            )
            for item in raw
        ]
```

C.3. Agent Module

Role. The agent module provides a uniform interface over heterogeneous reasoning systems. It hides implementation differences among direct model calls, lightweight scaffolds, and full agent runtimes behind a common `solve` interface. Given a task instance, the agent returns an `AgentResponse` containing the final prediction, interaction trajectory, termination reason, and execution metadata.

Model-harness system. The evaluated system is the model-harness pair $\mathcal{A}_{\theta,\mathcal{H}}$. The harness is not merely a prompt template: it determines the initial context, admissible actions, state initialization, parsing rules, error handling, tool exposure, and update logic. It also mediates the distinction between information placed in the model context and information maintained in the external runtime. For example, a prompt may describe available tools, while the actual tool registry, workspace state, and execution constraints are managed outside the model.

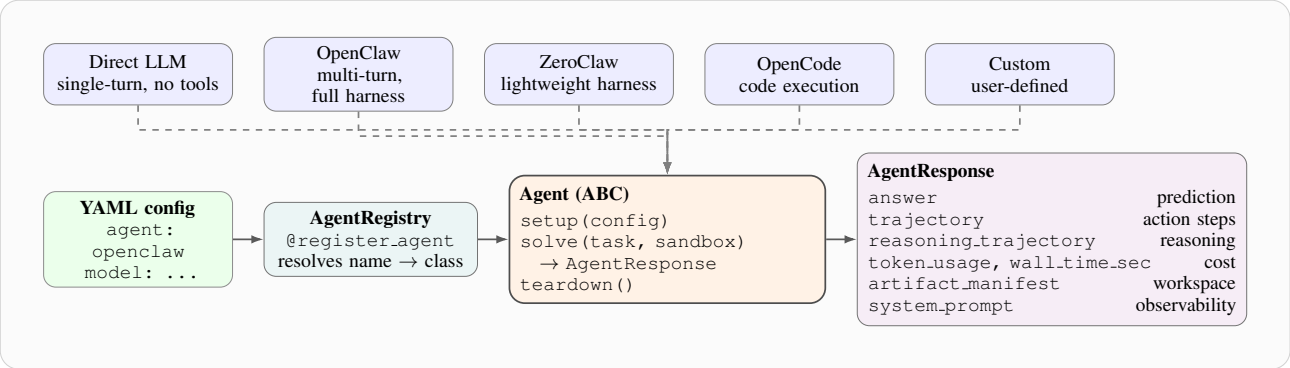


Figure 11. **Agent Module.** Concrete agent implementations (top) subclass the abstract `Agent` interface, which exposes `solve(task, sandbox)` → `AgentResponse`. The registry resolves an agent name from the YAML configuration at runtime. `AgentResponse` captures the full execution provenance: prediction, action trajectory, reasoning trace, cost metrics, workspace artifacts, and harness observability fields.

Direct and harnessed modes. AlphaDiana supports both direct and harnessed modes through the same external interface. In direct mode, the system performs a single model call with minimal task formatting, extracts an answer, and returns a degenerate one-turn trajectory. In harnessed mode, the system executes the multi-turn model-harness-environment loop defined in Section C.1. The harness may expose tools, manage persistent state, recover from invalid actions, incorporate environment feedback, and decide how observations enter later model contexts.

AgentResponse. The `AgentResponse` object records both the final prediction and the provenance of the interaction that produced it. At minimum, it contains the predicted answer $\hat{y}$, trajectory τ, and finish reason. It may also include token usage, wall-clock time, number of turns, number of tool calls, raw model responses, exposed tools, error states, workspace artifacts, and environment snapshots. These fields make it possible to analyze not only whether the final answer was correct, but also how it was produced.

Module boundary. The agent is separated from both the benchmark and the scorer. The benchmark provides the task input and reference object, but does not prescribe the agent's reasoning strategy. The scorer evaluates the prediction after generation, but does not participate in prompting, tool use, or recovery. This boundary is necessary for controlled cross-harness comparison: agents may differ in how they solve tasks, but they are evaluated against the same task and scoring interface.

Interface. Adding a new agent requires implementing:

`setup(config)`

for initializing the model client, harness configuration, prompt templates, tool registry, and runtime state; and

`solve(task, environment)` → `AgentResponse`

for executing a task and returning the prediction, trajectory, and metadata. Once registered, the agent can be selected directly from the YAML experiment specification.

Listing 2. Agent interface (`base.py`, verbatim) and a minimal custom implementation.

```
# --- base.py (verbatim) ----------------------------------
@dataclass
class AgentResponse:
    answer: Any                          # final prediction
    trajectory: list[dict] = ...        # action steps
    reasoning_trajectory: list[dict] = ... # reasoning trace
    token_usage: dict = ...             # cost metrics
    wall_time_sec: float = 0.0
    artifact_manifest: dict = ...       # workspace artifacts
    system_prompt: str = ""             # harness observability
    # ... (further provenance fields)
```

```
class Agent(ABC):
    @abstractmethod
    def setup(self, config: dict) -> None: ...

    @abstractmethod
    def solve(self, task: BenchmarkTask,
              sandbox: Optional[Any] = None) -> AgentResponse: ...

# --- custom implementation (minimal example) --------------
@register_agent("my_agent")
class MyAgent(Agent):
    def setup(self, config: dict) -> None:
        self.model = config["model"]

    def solve(self, task, sandbox=None) -> AgentResponse:
        answer = call_my_model(self.model, task.problem)
        return AgentResponse(answer=answer)
```

C.4. Environment Module

Role. The environment module provides the external execution substrate for agents that need to run code, access files, invoke tools, or interact with task-specific artifacts. It is optional for direct-model evaluation but becomes a first-class component in harnessed settings because it determines where actions are executed, what resources are available, what limits apply, and what isolation boundary separates agent behavior from the evaluation host.

Module boundary. The runner creates and manages the environment session, then passes it to the agent during task execution. The benchmark and scorer do not interact with the environment directly. This preserves the division of responsibilities: the benchmark specifies the task, the agent selects actions, the environment executes admissible actions, and the scorer evaluates the final prediction.

Session interface. An environment backend is exposed through a `SandboxSession` interface with the following:

$$\texttt{execute(command)} \rightarrow \texttt{ExecutionResult},$$

$$\texttt{upload(path, content)}, \quad \texttt{download(path)},$$

$$\texttt{reset()}, \quad \texttt{close()}, \quad \texttt{metadata()} \rightarrow \texttt{dict}.$$

These operations support command execution, file transfer, task-level cleanup, session split, and provenance reporting. Any backend implementing this interface is interchangeable in AlphaDiana.

State reset and reproducibility. The `reset()` operation is important for reproducibility. Agentic tasks often modify the workspace by creating files, installing packages, writing logs, editing repositories, or caching intermediate outputs. Without explicit reset semantics, later tasks may inherit hidden state from earlier tasks, making results order-dependent. AlphaDiana, therefore, treats reset behavior as part of the environment contract. A backend may implement reset by restoring a filesystem snapshot, clearing temporary directories, restarting services, or recreating task-local state.

ROCK backend. The default implementation uses a ROCK-based containerized backend. ROCK isolates agent actions from the evaluation host, confines command execution and file edits to a sandbox boundary, records resource profiles and image provenance, and supports retrieval of workspace artifacts after completion. This standardizes the execution substrate across runs and reduces variation due to local machine state, dependency drift, or uncontrolled filesystem differences. These properties are especially important for evaluating agents on tasks involving code execution, terminal interaction, repository modification, or artifact generation.

Listing 3. Sandbox interface (`base.py`, verbatim) and a minimal custom backend.

```
# --- base.py (verbatim) ----------------------------------
@dataclass
class ExecutionResult:
    exit_code: int
    stdout: str
```

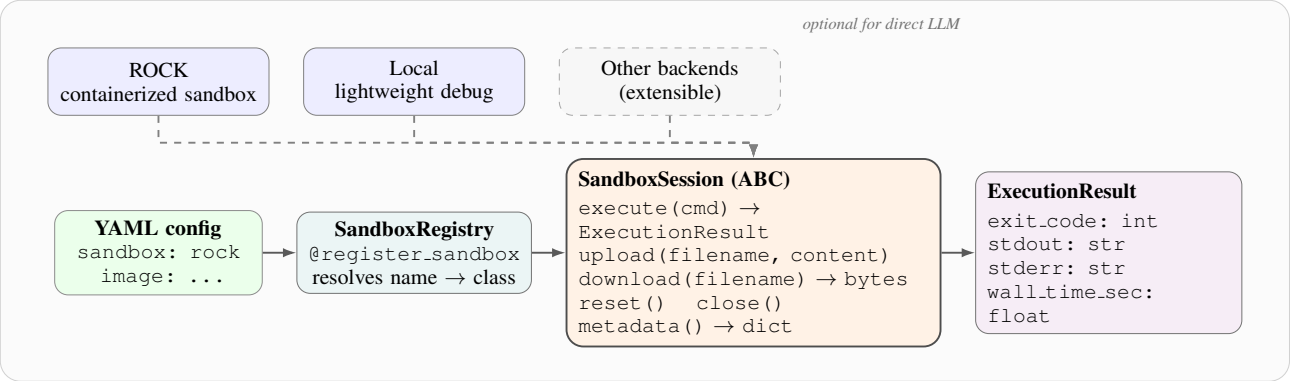


Figure 12. **Environment Module.** Backend implementations (top) subclass the abstract `SandboxSession` interface. The sandbox is optional: direct LLM evaluation requires no execution backend; harnessed agents (e.g., OpenClaw) use a ROCK-based containerized sandbox. The interface contract — `execute`, `upload`/`download`, `reset`, `close`, `metadata` — is the portability boundary that makes backends interchangeable.

```
    stderr: str
    wall_time_sec: float

class SandboxSession(ABC):
    @abstractmethod
    def execute(self, command: str) -> ExecutionResult: ...
    @abstractmethod
    def upload(self, filename: str, content: bytes) -> None: ...
    @abstractmethod
    def download(self, filename: str) -> bytes: ...
    @abstractmethod
    def close(self) -> None: ...
    def reset(self) -> None: ...          # clear state between tasks
    def metadata(self) -> dict: ...      # backend provenance

# --- custom backend (minimal example) --------------------
@register_sandbox("my_sandbox")
class MySandboxSession(SandboxSession):
    def execute(self, command: str) -> ExecutionResult:
        result = subprocess.run(command, shell=True, capture_output=True)
        return ExecutionResult(
            exit_code=result.returncode,
            stdout=result.stdout.decode(),
            stderr=result.stderr.decode(),
            wall_time_sec=0.0,
        )
    def upload(self, filename, content): ...
    def download(self, filename): ...
    def close(self): ...
```

C.5. Scorer Module

Role. The scorer module evaluates whether an agent's final prediction solves a task. Given the task input x, reference object y, prediction $\hat{y}$, and optionally the trajectory τ, the scorer returns a judgment

$$s = \mathcal{S}(x, y, \hat{y}, \tau).$$

The scorer is external to generation: it does not control prompts, tool use, environment interaction, recovery behavior, or answer generation. Its role begins only after the agent terminates. This separation ensures that correctness is determined by a shared verification rule rather than by harness-specific post-processing or self-judgment.

ScoreResult. The output is a `ScoreResult`, which contains a binary `correct` label, a scalar `score`, the expected

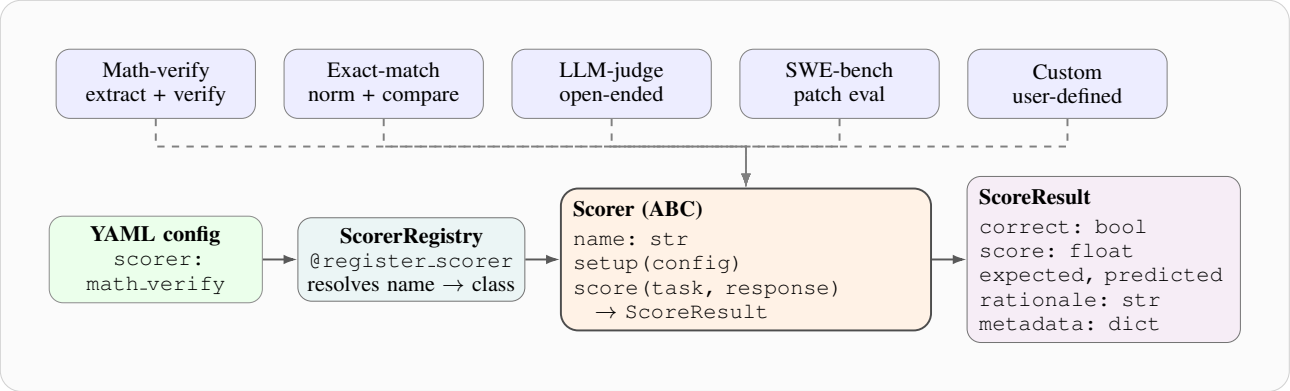


Figure 13. **Scorer Module.** Scorer implementations subclass the abstract `Scorer` interface, which maps a task and agent response to a `ScoreResult`. The scorer is fully decoupled from agent execution: it sees only the predicted answer and the reference, not how the prediction was produced.

reference value, the predicted value, a rationale, and metadata. The binary label supports accuracy-style metrics, while the scalar score supports partial credit or continuous-valued grading. Metadata may store normalization details, verifier outputs, judge responses, tolerance settings, error messages, or failure labels.

Built-in scoring strategies. AlphaDiana supports three built-in scoring strategies. Exact-match scoring compares normalized strings or categorical labels:

$$\texttt{correct} = \mathbf{1}[\mathrm{norm}(\hat{y}) = \mathrm{norm}(y)] \,.$$

Math-aware scoring first extracts the final answer and then applies symbolic, numeric, interval, unit, or equivalence checks:

$$\texttt{correct} = \mathbf{1}\big[\mathrm{verify}\big(\mathrm{extract}(\hat{y}), y\big)\big] \,.$$

LLM-as-judge scoring delegates the judgment to an independent judge model or judge pipeline,

$$\texttt{ScoreResult} = J_\phi(x, y, \hat{y}),$$

using a fixed rubric, prompt, model, and decoding configuration recorded in the scorer metadata. This strategy is used when deterministic matching is too brittle, when multiple acceptable responses exist, or when partial-credit rubrics are required.

Correctness versus system-level measurement. The scorer determines task-level correctness, while the runner separately records system-level measurements such as latency, token usage, number of turns, tool-call counts, timeout rate, crash rate, and completion rate. Keeping correctness separate from runtime behavior allows systems to be compared along multiple axes without conflating accuracy with cost or reliability.

Interface. Adding a new scorer requires implementing:

$$\texttt{score(task, response)} \rightarrow \texttt{ScoreResult}.$$

The `task` argument provides the input, reference object, identifiers, metadata, and attachments; the `response` argument provides the prediction, trajectory, and agent-side metadata. Once registered, the scorer can be selected from the YAML experiment specification without modifying the benchmark, agent, environment, or runner.

Listing 4. Scorer interface (`base.py`, verbatim) and a minimal custom implementation.

```python
# --- base.py (verbatim) ----------------------------------
@dataclass
class ScoreResult:
    correct: bool
    score: float
    expected: Any
    predicted: Any
    rationale: str = ""
    metadata: dict = field(default_factory=dict)
```

```python
class Scorer(ABC):
    @property
    @abstractmethod
    def name(self) -> str: ...

    def setup(self, config: dict) -> None: ...  # optional

    @abstractmethod
    def score(self, task: BenchmarkTask,
              response: AgentResponse) -> ScoreResult: ...

# --- custom implementation (minimal example) --------------
@register_scorer("my_scorer")
class MyScorer(Scorer):
    @property
    def name(self) -> str:
        return "my_scorer"

    def score(self, task, response) -> ScoreResult:
        correct = response.answer == task.ground_truth
        return ScoreResult(
            correct=correct, score=float(correct),
            expected=task.ground_truth,
            predicted=response.answer,
        )
```

C.6. Runner

Role. The runner is the orchestration layer of AlphaDiana. It instantiates the benchmark, agent, environment, scorer, and configuration; manages the environment lifecycle; dispatches tasks to the agent; invokes the scorer after generation; and persists all information needed for later analysis. The runner does not solve tasks or judge correctness directly. Instead, it enforces the evaluation protocol specified by $\mathcal{C}$.

Result record. For each task attempt, the runner writes an append-only result record. Abstractly, the record for attempt j on task i is

$$\rho_{i,j} = \big(x_i,\, y_i,\, \hat{y}_{i,j},\, s_{i,j},\, \tau_{i,j},\, z_{i,j}\big),$$

where $\tau_{i,j}$ is the trajectory produced by that attempt and $z_{i,j}$ denotes metadata collected from the benchmark, agent, environment, scorer, and run configuration. In single-sample evaluation, each task has one attempt, so we write ρ_i as shorthand for $\rho_{i,1}$. In implementation, records are stored as JSONL entries containing task identity, attempt identity, prediction, score, trajectory, cost and efficiency metrics, environment provenance, harness observability fields, artifacts, raw response payloads, and termination information.

Trajectory-aware analysis. This schema supports trajectory-aware evaluation. It preserves not only whether the agent solved the task, but also how the solution was produced, what resources were consumed, which tools were used, what feedback was observed, and why the run terminated. Downstream analyses can therefore compute accuracy, inspect failure modes, compare harness behavior, analyze tool-use patterns, and audit resource consumption without re-running the experiment.

Failure preservation. The runner also preserves both system-level and model-level failures, including a model API call failing, an invalid action occurring, a harness crashing, an environment operation times out, a budget is exhausted, or a scorer raises an exception. The runner writes records with all available task, trajectory, metadata, error, and finish-reason fields. This prevents failed tasks from being silently discarded and allows completion rate, timeout rate, crash rate, and other reliability statistics to be computed from the same result store as accuracy.

Run-level aggregation. Let $\mathcal{R} = \{\rho_{i,j}\}$ denote the saved attempt-level records for a run, where $\rho_{i,j}$ is the record for attempt j on task i. In single-sample evaluations, each task has one attempt, so ρ_i may be used as shorthand for $\rho_{i,1}$. Standard aggregate metrics include accuracy, mean score, completion rate, timeout rate, crash rate, mean latency, token usage, number of turns, and number of tool calls. These summaries are derived from the saved records rather than from transient in-memory state, so reports can be regenerated after the run finishes. For multi-sample evaluation, AlphaDiana also supports common

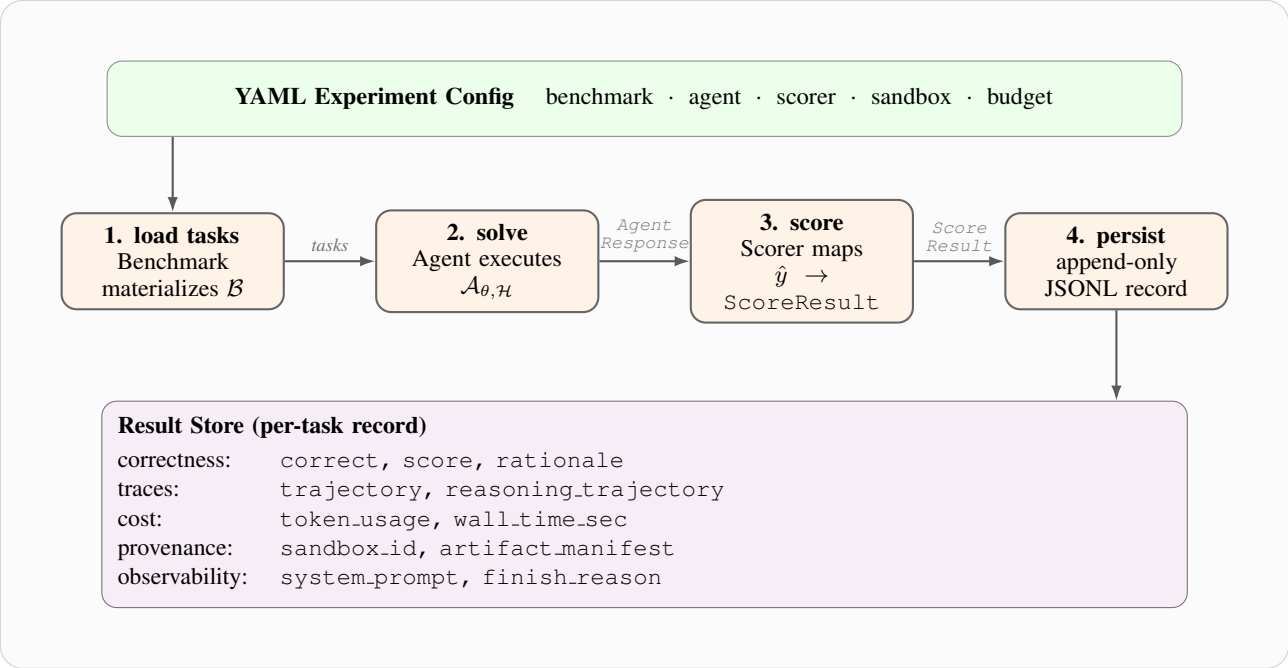


Figure 14. **Runner.** The runner instantiates all components from a YAML configuration and executes the four-stage pipeline: load tasks → solve → score → persist. Each per-task result record captures correctness, execution traces, cost metrics, sandbox provenance, and harness observability fields in a single append-only JSONL entry.

sampling-based metrics such as pass-rate and average correctness over repeated attempts for the same task. These metrics are computed by grouping attempt-level records by task identifier and aggregating the corresponding predictions, scores, trajectories, and metadata. This allows AlphaDiana to support both single-sample and multi-sample evaluation regimes without changing the underlying result schema.

Fine-grained provenance. Some analyses require information below the task level. For example, uncertainty estimation, calibration analysis, and token-level perplexity analysis require access to model log-probabilities when available. AlphaDiana therefore preserves raw model response payloads, including log-probability fields exposed by the model API, inside the response metadata. This enables downstream computation of decision uncertainty, confidence-correctness gaps, trajectory-level volatility, and related diagnostics.

Interface boundary. Overall, the runner coordinates modules without collapsing their responsibilities. It does not embed benchmark-specific prompting, agent-specific scoring, or scorer-specific execution logic. This boundary is what makes AlphaDiana experiments reproducible and comparable: different model-harness systems can be evaluated under the same benchmark, environment, scorer, and budget, while their trajectories and outcomes are stored in a common schema.

Listing 5. Minimal runner loop showing the four-stage pipeline. The actual `Runner` class handles concurrency, error recovery, and sandbox lifecycle; this excerpt shows the core per-task logic.

```
def run(config: ExperimentConfig, output_dir: str) -> None:
    # instantiate components from config
    benchmark = BenchmarkRegistry.get(config.benchmark_name)()
    agent     = AgentRegistry.get(config.agent_name)()
    scorer    = ScorerRegistry.get(config.scorer_name)()
    sandbox   = SandboxRegistry.get(config.sandbox_name)() \
                if config.sandbox_name else None

    benchmark.setup(config.benchmark_config)
    agent.setup(config.agent_config)
    scorer.setup(config.scorer_config)
    if sandbox: sandbox.setup(config.sandbox_config)

    store = ResultStore(output_dir, run_id=config.run_id)
```

```
    # four-stage pipeline
    for task in benchmark.load_tasks(config.benchmark_config): # 1. load
        session = sandbox.create_session() if sandbox else None
        response = agent.solve(task, session)                  # 2. solve
        result   = scorer.score(task, response)                # 3. score
        store.append(task, response, result)                   # 4. persist
        if session: session.close()
```

C.7. Reproduction

Compute resources. All AlphaDiana experiments were conducted on a local GPU cluster with 16 NVIDIA A100 GPUs, each with 80GB memory. The GPUs were used for model serving, trajectory generation, and harness-based evaluation. Since our study focuses on evaluation-time behavior rather than large-scale model training, the compute budget was mainly spent on repeated benchmark execution across models, harnesses, and sampling configurations.

The source code is provided in the anonymous repository: https://anonymous.4open.science/r/AlphaDiana-AEE4/.