CLAWGYM II: EXPLORING BLACK-BOX REINFORCEMENT LEARNING ON AGENT HARNESS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent harnesses have substantially improved performance on long-horizon tasks by coordinating model interactions with complex environments. However, reinforcement learning through such harnesses remains underexplored, as scaling such training to long-horizon agent tasks introduces fundamental challenges. In this work, we present a **unified black-box RL framework** for stable and scalable optimization of general agents through complex harnesses. We first build a sandbox-based execution infrastructure that isolates task environments and native harnesses within temporary sandboxes for large-scale concurrent rollouts. We then decouple policy optimization from opaque harness execution and place a serving proxy at the model boundary to capture faithful model calls, reconstruct structured training trajectories, and preserve training–inference consistency through **black-box token-in-token-out**. Crucially, rather than training separate harness-specific policies, we formulate each task–harness pair as a basic training instance and jointly optimize a shared policy from rollouts produced by heterogeneous harnesses. With Qwen3-30A3B jointly trained through **OpenClaw** and **Claude Code**, our model improves Pass@1 on ClawGym-Bench by 7.40 points under OpenClaw and 14.81 points under Claude Code, while improving PinchBench by 17.62 and 17.28 points, respectively. The same framework also delivers consistent gains on more challenging task settings from JobBench and OfficeQA. Overall, our results show that heterogeneous black-box harnesses can be jointly used as training interfaces for a single general agent policy, enabling effective and scalable optimization across diverse execution systems.

1 INTRODUCTION

Agent harnesses (Tang et al., 2026) have emerged as the foundational operating layer of autonomous agent systems, coordinating the interaction between large language models (Zhao et al., 2023) and their environments throughout the task-solving process. They typically integrate system prompts, tool interfaces, context management, workflow orchestration, and retry and recovery mechanisms into a unified runtime. Production-grade systems such as Claude Code (Anthropic, 2026), Codex (OpenAI, 2026a), and OpenClaw (OpenClaw, 2026) demonstrate that carefully designed harnesses can substantially improve agent performance on complex long-horizon tasks.

As these harnesses become increasingly capable, they are accelerating the development of general autonomous agents that operate across diverse real-world workspaces and solve tasks spanning software engineering (Merrill et al., 2026; Chowdhury et al., 2024), office applications (Patwardhan et al., 2025; Databricks, 2025), AI assistant workflows (Ye et al., 2026; Qwen Team & Data Team, 2026), and MCP-enabled tool use (Li et al., 2025; Bandi et al., 2026). Recent progress has been driven by inference-side advances in harness design, optimization, and self-evolution (Wang et al., 2026). Yet the benefits of increasingly capable harnesses depend not only on their design but also on how effectively the underlying model can leverage them. Even a strong model may perform poorly within a capable harness if it has not been suitably trained, motivating efforts, typically via reinforcement learning (RL) algorithms, to enable models to better exploit these agent harnesses.

A defining characteristic of modern harnesses is their inherent complexity: the design and implementation are sufficiently intricate. As a result, RL training algorithms targeting a harness can differ substantially from those used in fully textual Chen et al. (2025) or otherwise simple environ-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

ments Song et al. (2025; 2026). In simpler single-turn or lightweight agentic settings, the interaction process is explicit and directly observable, allowing RL to optimize well-defined model trajectories without accounting for complex hidden execution logic. With modern harnesses, by contrast, the mainstream approach Xu et al. (2026); Huang et al. (2026) is to treat the internal control flow and execution logic as opaque and to design a black-box RL algorithm. Indeed, black-box RL has become a key technical cornerstone for developing and leveraging capable harnesses for general agents Yu et al. (2026). Yet stable and scalable policy optimization for general agent tasks through such harnesses remains relatively underexplored in the existing literature. This work therefore focuses on the following research question:

How can black-box RL be performed through a harness to stably optimize agents?

To answer this question, we center our investigation on three fundamental challenges:

• *Scalable and stable infrastructure for black-box RL.* General agent tasks require dedicated, stateful environments during RL, and large-scale concurrent rollouts place substantial pressure on the execution infrastructure. For long-horizon tasks, runtime delays and failures can accumulate across interaction steps, making timely and reliable data generation difficult at scale.

• *Efficient and effective policy optimization from fragmented black-box traces*. Black-box harnesses expose fragmented, forked, and often redundant model calls rather than complete interaction trajectories, making these traces difficult to use directly for reinforcement learning. Moreover, opaque internal operations can further training–inference inconsistency, leading to unstable training.

• *Extensibility across heterogeneous harnesses*. Harnesses differ substantially in their interaction protocols, tool interfaces, context management, and execution workflows. A practical framework should support diverse harnesses with minimal system-specific adaptation and enable unified training across them, rather than optimizing policies for a single execution system.

To address these challenges, we develop a **unified black-box RL framework** for stably optimizing general agents through heterogeneous complex harnesses. First, to sustain large-scale concurrent environment execution during rollout, we encapsulate each task-specific environment together with its native execution harness inside a temporary sandbox that is provisioned on demand and destroyed upon completion. This provides an isolated workspace for every rollout while allowing execution to scale with available sandbox capacity. Second, we decouple policy optimization from harness execution by treating the harness as an opaque rollout engine and intercepting model calls only at the serving boundary. The harness therefore retains its native tool use, context management, retry, and orchestration logic without requiring reimplementation inside the RL pipeline. To transform fragmented black-box model calls into trainable interaction traces, we reconstruct their shared histories using a **prefix tree**. Training–inference consistency is further preserved through the **black-box token-in-token-out** mechanism (Gallouédec & Rasul, 2026). Finally, we make heterogeneous harnesses a first-class component of the training formulation. For each task, execution through a compatible harness defines a **task–harness pair**, which serves as the basic rollout and optimization unit. Task–harness pairs from different execution systems can be naturally mixed within the same training run, while all resulting learning signals jointly update a single shared policy. This formulation allows heterogeneous harnesses to share one training pipeline.

We instantiate the framework with two structurally distinct harnesses: **OpenClaw**, a general-purpose harness for workspace-grounded assistant tasks, and **Claude Code**, a mature harness for long-horizon coding and terminal interaction. Rather than training separate harness-specific policies, we jointly train Qwen3-8B and Qwen3-30A3B (Yang et al., 2025) using rollouts generated through both harnesses on ClawGym-SynData (Bai et al., 2026b), producing the unified ClawII-8B and ClawII-30A3B models. The same trained policy is then evaluated independently under each harness. For Qwen3-30A3B, joint black-box RL improves Pass@1 on ClawGym-Bench by 7.40 points under OpenClaw and 14.81 points under Claude Code, and improves PinchBench by 17.62 and 17.28 points, respectively. Furthermore, the same framework also produces improvements on more challenging tasks like JobBench (Li et al., 2026) and OfficeQA (Opsahl-Ong et al., 2026).

Our main contributions are summarized as follows:

- We establish a unified and extensible black-box RL formulation for training general agents directly through opaque execution harnesses, treating each task–harness pair as a basic training instance and enabling heterogeneous harnesses to jointly optimize a shared model.
- We develop the execution and optimization infrastructure required for stable and scalable black-box RL, including isolated sandbox-based rollouts, model-boundary interception, prefix-tree trajectory reconstruction, and black-box token-in-token-out.
- We validate the framework with OpenClaw and Claude Code, showing that a single jointly trained model consistently improves under both harnesses and across model scales and task categories, while the same framework further extends to challenging tasks.

2 PRELIMINARIES

2.1 PROBLEM FORMULATION FOR GENERAL AGENT TASKS

A general agent task consists of a user instruction together with an isolated, stateful environment in which the task is executed. The environment is initialized from a workspace and provides the execution context, including files, directories, documents, software repositories, running services, and other task-specific resources. To accomplish the task, the agent interacts with this environment over multiple turns through external tools, such as web search, file editing, code execution (Bai et al., 2026a), shell commands, or API calls, progressively transforming its state toward the desired goal. The outcome is reflected in the final environment state, typically through generated artifacts (e.g., documents, presentations, or reports) or successfully completed operations (e.g., fixing a software repository or configuring a system). We denote a task as $q = (u, \mathcal{W}_0)$, where u is the user instruction and $\mathcal{W}_0$ is the initial workspace, and use $\mathcal{E}_q$ to denote the associated execution environment initialized from $\mathcal{W}_0$, with tasks sampled from $q \sim \mathcal{D}$.

During execution, the agent alternates between observations and actions and produces a trajectory

$$\tau = (o_1, a_1, \ldots, o_T, a_T),$$

where o_t denotes the observation at step t and a_t denotes the corresponding agent action. The interaction terminates when the task is completed or execution stops, resulting in a final workspace state $\mathcal{W}_T$. This final state is then evaluated to obtain a rollout-level reward.

In practice, task evaluation typically follows two paradigms. Rule-based rewards verify the correctness of the final workspace through deterministic procedures, such as unit tests, program execution, or exact-match validation. Rubric-based rewards rely on a stronger evaluator (e.g., an LLM-as-a-Judge) to assess the overall quality of the generated artifact or final workspace according to task-specific evaluation criteria.

2.2 HARNESS-DRIVEN AGENT EXECUTION

Modern general-purpose agents are increasingly deployed through mature harnesses, such as OpenClaw, Claude Code, and Codex. Compared with traditional handcrafted agent loops based on the standard ReAct (Yao et al., 2022) paradigm, these harnesses (AgentScope-AI, 2026; HKUDS, 2026) provide a substantially richer execution abstraction between the language model and the external workspace. They integrate capabilities such as tool orchestration, context management, built-in skills, subagent delegation, and failure recovery into a unified runtime system, enabling agents to solve substantially more complex long-horizon tasks while abstracting away many low-level execution details from the model.

Under this execution paradigm, the model no longer interacts directly with the workspace. Instead, given a model action a_t, the harness executes the tools, updates the workspace and execution context, and returns the processed observation for the next decision. This interaction can be abstracted as

$$(\mathcal{W}_{t+1},\, o_{t+1},\, c_{t+1}) = \mathcal{H}(\mathcal{W}_t,\, c_t,\, a_t), \tag{1}$$

where $\mathcal{H}$ denotes the harness, $\mathcal{W}_t$ is the workspace state, c_t is the maintained execution context, and o_{t+1} is the observation returned to the model. Consequently, the interaction trajectory is deter-

mined jointly by the model and the harness rather than by the model alone, making the harness the fundamental execution abstraction for modern long-horizon agent systems.

2.3 Agentic Reinforcement Learning Algorithms

In conventional reasoning RL, such as mathematical reasoning, each rollout typically consists of a single model-generated response associated with an outcome reward. Agentic RL, in contrast, involves multi-turn interactions that interleave policy-generated actions with observations returned by the environment. Since these observations are not generated by the policy, they are masked out during optimization, ensuring that learning signals are applied only to policy-generated tokens.

Another characteristic of agentic RL is that a single rollout may yield multiple trainable trajectories, for example due to context-management mechanisms that split or restructure long-horizon interactions. While these trajectories may correspond to different segments of the same execution, the resulting environment state is evaluated only once. Consequently, trajectories originating from the same rollout share the same outcome reward.

3 Unified Black-Box RL through Complex Agent Harnesses

Our black-box RL formulation directly optimizes a trainable model through an unmodified and opaque agent harness, while decoupling policy optimization from harness execution. This section presents the infrastructure for scalable black-box rollouts, the recovery and optimization of training trajectories from captured model calls, and the extension to training across heterogeneous harnesses.

3.1 Infrastructure for Scalable Black-Box RL Execution

Black-box RL for general agent tasks requires large-scale concurrent execution of dedicated, stateful environments. Unlike stateless generation, each rollout may modify files, processes, tool states, and other task-specific resources over many interaction steps, making isolated execution essential. For each task q, we initialize a task-specific environment from its initial workspace $\mathcal{W}_0$ and launch the selected harness inside a temporary sandbox. The sandbox provides an isolated workspace and the runtime dependencies required by the harness, preventing interference across concurrent rollouts. Each sandbox is provisioned at rollout start and released after completion, allowing execution to scale with available sandbox capacity. In addition to isolating task states, each sandbox provides the external capabilities required during rollout. Beyond the basic tools supplied natively by the selected harness, common capabilities (*e.g.,* web search), together with task-specific tools, are exposed through Model Context Protocol (MCP) servers launched inside the sandbox. Standardized MCP interfaces ensure reliable access to these services throughout execution and allow new capabilities to be integrated without modifying the native harness workflow.

We decouple policy optimization from harness execution. The training engine is responsible for policy optimization, while the inference engine serves the policy during rollout. The harness drives the interaction inside the sandbox and retains control over tool use, context management, retries, and environment interaction. This separation avoids reimplementing harness-specific control flow in the training pipeline and enables different black-box harnesses to be integrated with minimal adaptation.

Although the internal control flow of the harness is inaccessible, model behavior remains observable at the model-serving boundary. Every model-generated action is produced through a model request that crosses this boundary. We therefore place a serving proxy at the boundary as the model endpoint used by the harness. For each request, the proxy invokes the rollout policy, returns the generated response in the protocol expected by the harness, and records the input tokens, generated tokens, rollout log-probabilities, and associated task metadata. This boundary-level capture provides the model-side information required for optimization without instrumenting the internal harness logic.

As illustrated in Figure 1, each rollout launches the task environment and harness inside a sandbox, after which all model requests pass through the serving proxy. Upon completion, the verifier evaluates the final workspace state and produces a rollout-level reward. The captured model-call records and reward are passed to the training pipeline, where the calls are reconstructed into trainable multi-turn trajectories, as described in Section 3.2. The training engine then updates the policy model and synchronizes the parameters to the inference engine for the next iteration.

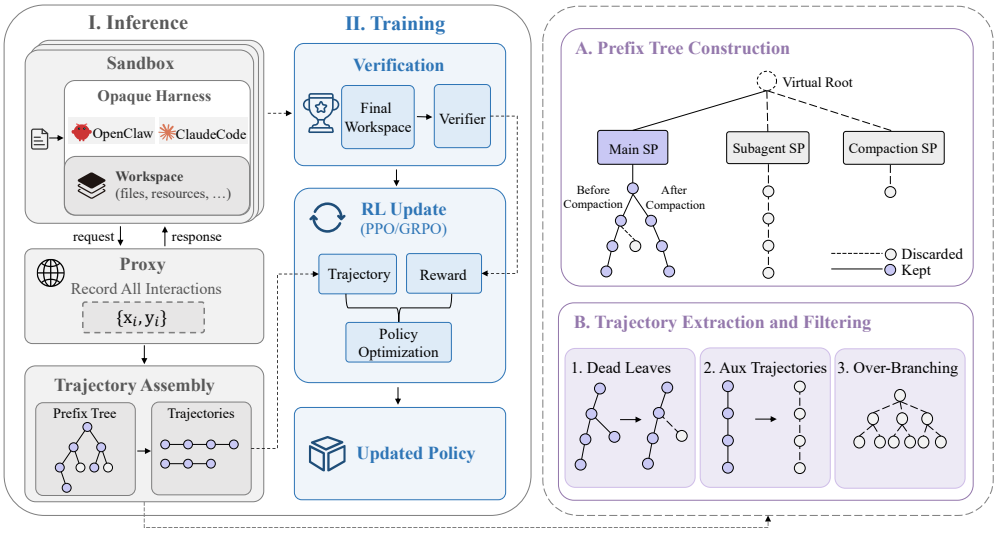


Figure 1: Overview of our black-box RL framework for optimizing agents through harnesses.

3.2 Bridging Black-Box Harness Execution and Policy Optimization

The serving proxy captures the model-side records required for optimization, but these records must first be transformed into structured multi-turn training trajectories. We organize the captured calls into prefix trees, recover trainable multi-turn trajectories from the resulting structure, and preserve training–inference consistency throughout policy optimization.

3.2.1 Prefix Tree Construction

The model calls captured during a black-box rollout are fragmented, forked, and potentially redundant. Processing each call as a separate training sample can repeat computation over shared histories, even when the policy loss is restricted to tokens generated by that call. We therefore organize the model calls captured from each rollout into a *rollout-level prefix tree*, which reconstructs their shared interaction structure and provides the training representation over which optimization is performed.

Concretely, consider a rollout that produces model calls $\mathcal{C} = \{(x_i, y_i)\}_{i=1}^{m}$, where x_i is the input context of the i-th call and y_i is the corresponding model response. Within an uninterrupted interaction segment, later inputs extend histories established by earlier calls. Context compaction or subagent execution may start a new segment from a shortened or alternative context, after which subsequent calls continue to extend the newly established history. We construct a tree T rooted at the initial task prompt and attach each call to the existing node whose accumulated history forms the longest prefix of x_i. By comparing the input of a child call with the accumulated parent history and the recorded parent response, we also recover the intervening non-model content introduced by the harness, including tool outputs and other environment feedback.

A leaf is a call that no later call extends, and each root-to-leaf path defines a candidate multi-turn trajectory. Mature harnesses may introduce multiple leaves through context management or subagent scheduling, which start new interaction branches instead of continuing the current one. By representing shared interaction histories with common tree nodes, the prefix tree stores each shared prefix only once while preserving distinct continuations as separate branches. During optimization, each retained sampled-token occurrence contributes to the policy loss only once across the extracted trajectories, even when it belongs to multiple root-to-leaf paths, preventing more highly branched rollouts from receiving disproportionate optimization weight.

3.2.2 Trajectory Extraction and Filtering

The prefix tree represents the model-call structure exposed during the task-solving process, and tracing a valid leaf back to the root recovers a candidate multi-turn training trajectory. However, not

every leaf corresponds to the completion of the main task-solving process, as some arise from fault handling, context management, or subagent processes. We filter the resulting trajectories to retain those suitable for training.

Removing the Dead Leaves. A single turn may be generated multiple times when the inference server retries a request or the harness regenerates an invalid response, as OpenClaw does after a malformed tool call. Superseded attempts receive no further continuation but remain in the prefix tree as dead leaves. We partition the candidate trajectories into interaction segments delimited by context-compaction points and, within each segment, retain the leaf with the longest valid continuation. Shorter sibling trajectories terminating at retry-induced dead leaves are discarded.

Discarding Over-Branching Tasks. When the prefix tree of a rollout branches into an excessive number of leaves, the rollout has typically entered repeated or failed generation rather than legitimate interaction branching, and the resulting trajectories provide little useful training signal. Once the number of leaves exceeds a preset threshold, we discard the corresponding rollout, sacrificing a small amount of data to prevent corrupted signals from entering the gradient update.

Excluding Auxiliary Trajectories. Subagent and compaction trajectories are excluded because their roles in the harness differ from the role of the main task-solving trajectory and their contributions are not directly aligned with the rollout-level task reward. Broadcasting the same terminal reward to these auxiliary interactions would introduce ambiguous credit assignment and noisy optimization signals, potentially destabilizing training. We therefore optimize only the main-agent trajectories and leave effective use of auxiliary interactions to future work.

3.2.3 Training–Inference Consistency

During RL, the policy loss must be computed on the exact tokens generated during rollout, with each token conditioned on the same context used at sampling time, ensuring that the sequence presented to the training engine matches the one actually sampled by the policy. In the black-box setting, this requirement is complicated by harness-side transformations, such as tool-call normalization and assistant-message re-serialization, which may alter the representation of model outputs before they re-enter subsequent interactions. Simply re-tokenizing the reconstructed root-to-leaf trajectories would encode the harness-transformed model responses, producing token sequences that may differ from those originally sampled during rollout.

To keep the trained sequence identical to the generated one, we adopt the *black-box token-in-token-out* discipline that maintains two decoupled views of every model call. The training view retains the exact input and generated tokens recorded at the inference boundary. The tokens generated by the inference engine are grafted directly onto the prefix tree and constitute the sole source of training data while the structured text that the harness receives is decoded from those same tokens purely for the harness to act on, and is never encoded back into the trained trajectory. Once a tool call is completed, the resulting environment response is encoded into a token sequence exactly as presented in the next model request and appended to the prefix tree. The two views thus serve different consumers: the harness drives the interaction from the decoded text, while training reads the generated tokens verbatim. Regardless of how the harness reformats or standardizes what it displays and feeds into later turns, those edits stay on its own view and cannot perturb the token record, so the sequence handed to the training engine is by construction identical to the one the policy model sampled.

3.3 Joint Training across Heterogeneous Harnesses

Harnesses can differ substantially in their interaction protocols, tool interfaces, context-management strategies, and internal control flows. A central goal of our framework is therefore not merely to support multiple harnesses independently, but to allow their rollout data to participate in a common optimization process while preserving the semantics of each execution system.

Our architecture naturally enables this capability because harness-specific execution is isolated from policy optimization. Every harness communicates with the policy through the same model-serving boundary, and its model calls are transformed into the same training representation before enter-

ing the optimization pipeline. Consequently, the policy optimizer does not need to reproduce or explicitly model the internal logic of any particular harness.

For each task q with environment $\mathcal{E}_q$, we construct separate training instances $(q, \mathcal{E}_q, H_k)$ by pairing the task with different compatible harnesses H_k. Since the environment is uniquely determined by the task, we refer to each instance as a task–harness pair. Instances associated with different harnesses are randomly mixed within each training batch and jointly processed through the shared training pipeline. Importantly, rollout statistics that depend on relative comparison are computed within each task–harness group rather than across harnesses. Executions of the same task through different harnesses may induce substantially different interaction trajectories, exploration patterns, and reward distributions. Keeping these groups separate prevents harness-dependent execution characteristics from contaminating relative credit assignment, while still allowing their resulting gradients to be aggregated in the same policy update.

This formulation makes heterogeneous-harness training a direct consequence of the unified black-box interface rather than a harness-specific modification to the RL algorithm. At evaluation or deployment time, the resulting shared policy can be placed behind any supported harness without requiring a separate harness-specific model.

4 Experiments

We evaluate the effectiveness and generality of our unified black-box RL framework. We first report the overall performance of a single policy jointly trained through OpenClaw and Claude Code, followed by a comparison with single-harness training to assess the effectiveness of joint training, and further evaluation on JobBench and OfficeQA to examine framework extensibility.

4.1 Overall Performance

4.1.1 Experimental Settings

Evaluation Datasets and Metrics. We evaluate all models using Pass@1 on ClawGym-Bench (Bai et al., 2026b) and PinchBench (PinchBench, 2026) under a hybrid protocol that combines code-based verification with rubric-based judgment. On ClawGym-Bench, tasks whose verifier consists solely of code checks are scored by that verifier alone, whereas tasks whose verifier combines code checks with rubrics are scored as a weighted sum of the two, with weights 0.7 and 0.3, respectively. For PinchBench, we take the task set released on April 10, 2026, discard the multimodal tasks, and retain 30 tasks, each scored with the verifier provided by the benchmark. All rubric-based judgments use GPT-5.4 (OpenAI, 2026b), with the prompt in Appendix A.

Evaluated Models. We use the Qwen3 series (Yang et al., 2025) as baselines. We jointly train Qwen3-8B and Qwen3-30A3B through OpenClaw (OpenClaw, 2026) and Claude Code (Anthropic, 2026) on ClawGym-SynData (Bai et al., 2026b) using GRPO, yielding ClawII-8B and ClawII-30A3B. Each trained model is evaluated under both harnesses with a maximum context length of 64K tokens, while longer trajectories are handled by the context management of the harness.

4.1.2 Results

Table 1 shows the performance of our black-box RL framework. We make the observations:

• *Heterogeneous-Harness Black-Box RL Delivers Substantial Gains.* Joint black-box RL consistently improves the corresponding Qwen3 initialization under both harnesses. On ClawGym-Bench, ClawII-30A3B improves over Qwen3-30A3B by 7.40 points under OpenClaw and 14.81 points under Claude Code. Similar gains are observed on PinchBench, where the improvements reach 17.62 and 17.28 points, respectively, showing strong transfer across benchmarks.

• *A Shared Policy Performs Strongly across Harnesses.* The same jointly trained ClawII model performs strongly when evaluated through either OpenClaw or Claude Code, without requiring separate harness-specific checkpoints. ClawII-30A3B achieves 52.51 and 51.87 on ClawGym-Bench under the two harnesses, respectively. It also outperforms the substantially larger Qwen3-235A23B on PinchBench under both evaluation settings, despite using far fewer parameters.

Table 1: Performance comparison of different models on ClawGym-Bench and PinchBench. The best and second-best Pass@1 results within each group are highlighted in **bold** and underlined.

Model	Pinch-Bench	ClawGym-Bench						
		Product. & Collab.	Systems & Auto.	Analysis & Reason.	Content & Domain	Planning & Knowl.	Software Dev.	Avg.
OpenClaw as Harness								
Qwen3-8B	54.50	37.46	29.06	30.40	41.12	44.47	30.69	35.02
Qwen3-32B	49.40	40.68	36.21	37.84	47.16	49.29	33.11	40.32
Qwen3-30A3B	55.60	42.47	42.09	45.04	51.95	45.98	46.24	45.11
Qwen3-235A23B	60.60	**53.66**	**52.27**	47.18	**61.39**	**67.33**	49.23	**54.48**
ClawII-8B	64.92	44.57	46.94	42.99	49.96	51.71	42.61	46.09
ClawII-30A3B	**73.22**	52.62	50.15	**53.74**	52.14	52.93	**55.94**	52.51
Claude Code as Harness								
Qwen3-8B	27.40	16.19	13.94	20.26	25.37	23.62	14.65	18.54
Qwen3-32B	53.27	33.34	25.63	27.68	33.03	27.85	28.30	29.37
Qwen3-30A3B	54.14	38.75	33.06	32.99	45.43	41.62	32.26	37.06
Qwen3-235A23B	62.47	41.74	40.53	43.70	50.35	49.03	54.90	45.59
ClawII-8B	61.21	41.27	38.19	41.65	49.33	44.61	39.37	42.05
ClawII-30A3B	**71.42**	**47.84**	**45.59**	**48.88**	**57.05**	**56.93**	**62.93**	**51.87**

• *Consistent Improvements across Task Categories.* The gains are broad rather than concentrated in only a small subset of tasks. Both ClawII-8B and ClawII-30A3B improve over their corresponding Qwen3 baselines across all six ClawGym-Bench categories under both OpenClaw and Claude Code. This consistency indicates that joint harness training benefits a diverse range of agent capabilities rather than only a narrow task type or domain.

4.2 Comparison with Single-Harness Training

To validate the effectiveness of our joint heterogeneous-harness training, we compare it against training through either harness alone and examine whether mixing heterogeneous harnesses compromises per-harness performance. Starting from Qwen3-30A3B, we perform GRPO training under three settings: OpenClaw only, Claude Code only, and joint training with both harnesses.

As described in Section 3.3, for each task, the joint setting instantiates separate task–harness pairs through OpenClaw and Claude Code. Task–harness pairs from both systems are randomly mixed within the same training batch, while GRPO grouping and relative-advantage normalization are performed independently for each pair. Both the single-harness and joint-training settings use the same rollout configuration of 32 task–harness instances with 8 rollouts per instance.

As shown in Figure 2, joint training achieves training rewards comparable to the corresponding single-harness runs. Under OpenClaw, it gradually closes the initial gap toward the end of training, while under Claude Code its training reward remains on par with or higher than the Claude-Code-only model. A similar pattern is observed in downstream evaluation, where the jointly trained model matches or slightly outperforms the corresponding single-harness models under both harnesses.

Overall, jointly training through heterogeneous harnesses introduces no evident performance degradation compared with training each harness independently. These results demonstrate that learning signals from heterogeneous harnesses can be jointly leveraged to optimize a shared policy model within our unified black-box RL framework, without evident performance degradation under either execution system.

4.3 Training on More Challenging Tasks

To further evaluate the generality of our unified framework, we extend black-box RL to more challenging and structurally diverse task settings from JobBench (Li et al., 2026) and OfficeQA (Opsahl-Ong et al., 2026). JobBench targets professional workflows over heterogeneous workspaces con-

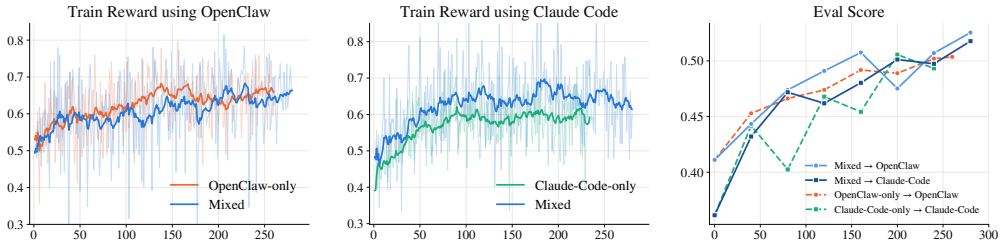
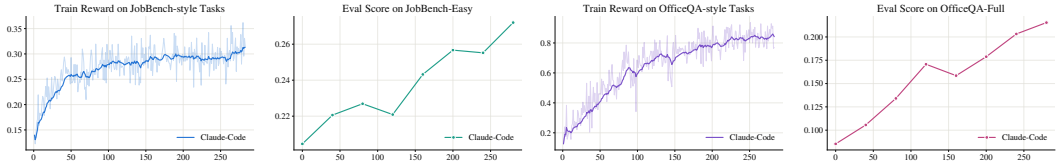


Figure 2: Comparison between single-harness and mix-harness RL training.

taining diverse file formats, including images, databases, and native Office files, requiring agents to interpret and integrate information across multiple sources to complete complex tasks. OfficeQA instead emphasizes answer-centric reasoning over a large document corpus, where agents must retrieve relevant evidence, perform multi-step analysis or computation, and produce a verifiable final answer rather than a workspace artifact. Together, the two benchmarks cover substantially different forms of complex agent interaction, from artifact-oriented workspace execution to document-grounded analytical reasoning.

Following their respective task formats, we synthesize **JobBench-style** and **OfficeQA-style** training tasks and perform black-box RL on Qwen3-30A3B with Claude Code as the rollout harness. Under our unified framework, a new task can be directly incorporated into training by simply formulating it as a triplet of an instruction, an initialized workspace, and a verifier, without requiring task-specific modifications to the RL pipeline. As shown in Figures 3a and 3b, evaluation performance improves from 20.46 to 27.20 on JobBench-Easy and from 8.53 to 21.54 on OfficeQA-Full. Training rewards also maintain clear and stable upward trends throughout optimization. These results demonstrate that our framework is not tied to a particular task format or evaluation paradigm: the same unified black-box RL pipeline can be readily extended to substantially different and more demanding task distributions, while continuing to deliver effective and stable policy optimization.



(a) JobBench-style tasks.

(b) OfficeQA-style tasks.

Figure 3: Training dynamics with Claude Code on (a) JobBench-style and (b) OfficeQA-style tasks.

5 Conclusion

In this paper, we present a unified black-box RL framework for training general agents directly through complex and opaque harnesses. Our framework decouples policy optimization from harness execution, supports scalable sandbox-based rollouts, reconstructs fragmented model calls into trainable trajectories, and preserves training–inference consistency. Moreover, by treating each task–harness pair as a basic training instance, it enables heterogeneous harnesses to jointly optimize a single shared policy. Experiments with OpenClaw and Claude Code demonstrate that joint heterogeneous-harness training consistently improves Qwen3 models under both execution systems without evident degradation compared with training each harness independently. With Qwen3-30A3B, ClawII improves Pass@1 on ClawGym-Bench by 7.40 points under OpenClaw and 14.81 points under Claude Code, with similarly strong gains on PinchBench. These improvements remain consistent across model scales and diverse task categories, showing that a shared policy can effectively absorb learning signals from heterogeneous execution systems. The same framework also extends effectively to more challenging task settings from JobBench and OfficeQA. Overall, our results suggest that complex agent harnesses can serve not only as inference-time execution systems, but also as unified training interfaces for scalable reinforcement learning of general agents.

AI USE STATEMENT

Generative AI tools, including large language models (LLMs), were used in this work for both manuscript preparation and synthetic data generation. For the manuscript, LLMs assisted with rewriting, proofreading, and stylistic refinement to improve clarity, fluency, grammatical accuracy, and overall presentation. AI-assisted revisions were subsequently reviewed to ensure that they faithfully reflected the intended technical meaning. LLMs were also incorporated into the data-generation pipeline to produce synthetic training data and to evaluate model outputs as judges. In particular, they were used to generate agent interaction trajectories and to construct and filter training examples used in our experiments. The specific models, generation procedures, filtering criteria, and related experimental settings are documented in the corresponding methodology and experimental sections of the paper. The authors take full responsibility for the accuracy, integrity, and final content of this work, including all text, datasets, claims, analyses, and other research artifacts produced with the assistance of generative AI.

REFERENCES

AgentScope-AI. QwenPaw: Your personal ai assistant. https://github.com/agentscope-ai/QwenPaw, 2026. GitHub repository. Accessed: 2026-04-29.

Anthropic. Claude Code: Ai-powered coding assistant for developers. https://claude.com/product/claude-code, 2026. Product page. Accessed: 2026-04-29.

Fei Bai, Yingqian Min, Beichen Zhang, Zhipeng Chen, Xin Zhao, Lei Fang, Zheng Liu, Zhongyuan Wang, and Hongteng Xu. Towards effective code-integrated reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 30022–30030, 2026a.

Fei Bai, Huatong Song, Shuang Sun, Daixuan Cheng, Yike Yang, Chuan Hao, Renyuan Li, Feng Chang, Yuan Wei, Ran Tao, et al. Clawgym: A scalable framework for building effective claw agents. *arXiv preprint arXiv:2604.26904*, 2026b.

Chaithanya Bandi, Razvan-Gabriel Dumitru, Ben Hertzberg, Divyansh Agarwal, Geobio Boo, Tejas Polakam, Sami Hassaan, Jeff Da, HiJae Kim, Vipul Gupta, et al. Mcp-atlas: A large-scale benchmark for tool-use competency with real mcp servers. *arXiv preprint arXiv:2602.00933*, 2026.

Zhipeng Chen, Yingqian Min, Beichen Zhang, Jie Chen, Jinhao Jiang, Daixuan Cheng, Wayne Xin Zhao, Zheng Liu, Xu Miao, Yang Lu, et al. An empirical study on eliciting and improving r1-like reasoning models. *arXiv preprint arXiv:2503.04548*, 2025.

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing swe-bench verified, August 2024. URL https://openai.com/index/introducing-swe-bench-verified/.

Databricks. Officeqa: A grounded reasoning benchmark, 2025.

Quentin Gallouédec and Kashif Rasul. Agentic rl: Token-in, token-out done right, May 2026. URL https://huggingface.co/blog/huggingface/tito. Hugging Face Blog.

HKUDS. NanoBot: The ultra-lightweight personal ai agent. https://github.com/HKUDS/nanobot, 2026. GitHub repository. Accessed: 2026-04-29.

Liangmeng Huang, Qingchuan Li, Hongwei Xue, Shilin Yan, and Dressage Contributors. Dressage: Scalable RL for any agent and any sandbox. https://github.com/Accio-Lab/Dressage, 2026.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, et al. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. *arXiv preprint arXiv:2510.25726*, 2025.

Yuetai Li, Yichen Feng, Zhangchen Xu, Zixian Ma, Kaiyuan Zheng, Fengqing Jiang, Xinghua Sun, Rulin Shao, Zichen Chen, Yue Huang, Xinyang Han, Brian Lee, Kayla Xu, Shenglai Zeng, Hang Hua, Xiangliang Zhang, Basel Alomair, Ranjay Krishna, Luke Zettlemoyer, Pang Wei Koh, Bhaskar Ramasubramanian, Luyao Niu, Xiang Yue, and Radha Poovendran. Jobbench: Aligning agent work with human will, 2026. URL https://arxiv.org/abs/2605.26329.

Mike A Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Jenia Jitsev, Marianna Nezhurina, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Kumar Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Kwesi Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=a7Qa4CcHak.

OpenAI. Codex: Ai coding partner from openai. https://openai.com/codex/, 2026a. Product page. Accessed: 2026-04-29.

OpenAI. Introducing GPT-5.4. https://openai.com/index/introducing-gpt-5-4/, March 2026b. Official release announcement. Accessed: 2026-04-29.

OpenClaw. OpenClaw: Personal ai assistant. https://github.com/openclaw/openclaw, 2026. GitHub repository. Accessed: 2026-04-29.

Krista Opsahl-Ong, Arnav Singhvi, Jasmine Collins, Ivan Zhou, Cindy Wang, Ashutosh Baheti, Owen Oertell, Jacob Portes, Sam Havens, Erich Elsen, Michael Bendersky, Matei Zaharia, and Xing Chen. Officeqa pro: An enterprise benchmark for end-to-end grounded reasoning, 2026. URL https://arxiv.org/abs/2603.08655.

Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, Grace Kim, Michele Wang, Olivia Watkins, Simón Posada Fishman, Marwan Aljubeh, Phoebe Thacker, Laurance Fauconnet, Natalie S. Kim, Patrick Chao, Samuel Miserendino, Gildas Chabot, David Li, Michael Sharman, Alexandra Barr, Amelia Glaese, and Jerry Tworek. Gdpval: Evaluating ai model performance on real-world economically valuable tasks, 2025. URL https://arxiv.org/abs/2510.04374.

PinchBench. PinchBench: Real-world benchmarks for ai coding agents. https://github.com/pinchbench/skill, 2026. GitHub repository. Accessed: 2026-04-29.

Qwen Team and Alibaba Group Data Team. QwenClawBench: Real-user-distribution benchmark for openclaw agents, April 2026. URL github.com/SKYLENAGE-AI/QwenClawBench.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*, 2025.

Huatong Song, Lisheng Huang, Shuang Sun, Jinhao Jiang, Ran Le, Daixuan Cheng, Guoxin Chen, Yiwen Hu, Zongchao Chen, Yiming Jia, et al. Swe-master: Unleashing the potential of software engineering agents via post-training. *arXiv preprint arXiv:2602.03411*, 2026.

Xinyu Tang, Han Peng, Guoxin Chen, Yuze Shi, Zitao Su, Peiyu Liu, Wayne Xin Zhao, Yawen Li, and Zhe Xue. Agent systems with harness engineering, 2026. URL https://openreview.net/pdf?id=nM5tDHrQsx.

Ruhan Wang, Yucheng Shi, Zongxia Li, Zhongzhi Li, Yue Yu, Junyao Yang, Kishan Panaganti, Haitao Mi, Dongruo Zhou, et al. Harness handbook: Making evolving agent harnesses readable, navigable, and editable. *arXiv preprint arXiv:2607.13285*, 2026.

Binfeng Xu, Hao Zhang, Shaokun Zhang, Songyang Han, Mingjie Liu, Jian Hu, Shizhe Diao, Zhenghui Jin, Yunheng Zou, Michael Demoret, et al. Polar: Agentic rl on any harness at scale. *arXiv preprint arXiv:2605.24220*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, et al. Claw-eval: Toward trustworthy evaluation of autonomous agents. *arXiv preprint arXiv:2604.06132*, 2026.

Xiao Yu, Baolin Peng, Ruize Xu, Hao Zou, Qianhui Wu, Hao Cheng, Wenlin Yao, Nikhil Singh, Zhou Yu, and Jianfeng Gao. Openforgerl: Train harness-native agents in any environment. *arXiv preprint arXiv:2607.21557*, 2026.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2), 2023.

A Appendix

```
You are a strict rubric-based evaluator. Use only the user prompt
    content. Do not call tools, browse, inspect files, or ask for more
    context. Your response may include concise analysis, but it must end
    with exactly one standalone JSON object with keys `scores` and `notes
    `.

You are grading an agent result.

You must not call, request, or simulate any tools. Do not browse, list
    directories, open files, inspect the workspace, or ask for more
    context. Grade only from the task, final output files, optional
    transcript evidence, and rubrics included in this prompt.

Before giving the final judgment, provide concise analysis explaining
    how the final outputs satisfy or fail each rubric criterion.

Your final judgment must end with exactly one standalone JSON object and
     nothing else after it. The final JSON object must contain exactly
    two keys: `scores` and `notes`.

- `scores` must map every rubric id (`criterion_1`, `criterion_2`, ...)
    to one numeric score chosen from that rubric's allowed score anchors.
- `notes` must be a concise string summarizing the main reasons for the
    assigned scores.

Do not include any overall score in the final JSON. Do not compute or
    report the final aggregated score. Score aggregation will be handled
    separately by post-processing.

## User Task
{{USER_TASK}}
```

```
## Final Output Files
{{FINAL_OUTPUT_FILES}}

## Additional Changed Workspace Files
{{ADDITIONAL_CHANGED_WORKSPACE_FILES}}

## Transcript
{{TRANSCRIPT_OPTIONAL}}

## Rubric
{{RUBRIC}}
```