OARS: ON POST-TRAINING FOR COST-EFFICIENT HETEROGENEOUS MULTI-AGENT SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Frontier large language models (LLMs) are powerful agentic problem solvers, but deploying them at every step of a long-horizon task incurs substantial inference cost. Such tasks involve many rounds of LLM reasoning and environment interaction, yet not every step requires the same level of model capability. We introduce OARS (Orchestrator-Aware Role Specialization), which reduces reliance on expensive large models by organizing specialized small models to perform delegated work. Specifically, a frozen large-model orchestrator decomposes and delegates tasks to small-model workers through a unified work-order protocol that governs information isolation and exchange. Workers return only concise, task-relevant findings rather than their full execution histories, thereby reducing the orchestrator's computational workload and context growth. We post-train small-model workers within the complete multi-agent system. A central challenge is the *free-rider problem*: workers can be rewarded for collective success regardless of their own contribution, as a strong orchestrator or other workers may compensate for unhelpful worker reports. We address this with contribution-aware training that uses worker-level progress rewards to distinguish individual contributions from collective success. Our hierarchical credit normalization further prevents episodes with more workers or longer trajectories from receiving disproportionate training weight. On BrowseComp-Plus, OARS—with 8B workers assisting 230–550B-parameter models—improves accuracy by 6.3 percentage points on average while being up to 10.6× more cost-efficient than standalone large-model agents. On DR-256, OARS improves accuracy by 19.9 percentage points on average while achieving up to a 4.2× reduction in cost per correct answer. OARS further extends to agentic coding, cutting cost per resolved task by 66.5% on SWE-bench Verified while maintaining accuracy. We will release the code upon publication.

1 INTRODUCTION

Large language models (LLMs) have demonstrated remarkable capabilities in solving complex problems. By using tools to interact with external environments (Yao et al., 2023), LLM agents further extend these capabilities to a broader range of challenging tasks, including scientific discovery (Hubert et al., 2026; Novikov et al., 2025; OpenAI, 2026), web search (Nakano et al., 2021; Jin et al., 2025; OpenAI, 2025), and software engineering (Yang et al., 2024; Wang et al., 2025b; Antoniades et al., 2025). However, repeated interactions and growing contexts create substantial inference workloads for LLM agents. For example, Kimi K2.6 made over 4,000 tool calls during a single coding task (Moonshot AI, 2026), while Claude Opus 4.6's multi-agent BrowseComp (Wei et al., 2025a) evaluation averaged over 6 million tokens per task (Anthropic, 2026b). Together, these long trajectories and massive token consumption translate into substantial inference costs.

The structure of agentic tasks offers an opportunity to reduce these costs. A task may require a powerful model, but not at every step. Much of an agent's work involves gathering and filtering information to support a smaller number of consequential decisions. Delegating this work to multiple sub-agents, however, can increase total cost, as multi-agent configurations may incur over $5\times$ the token usage of single-agent execution (Hadfield et al., 2025; Anthropic, 2026b). This motivates assigning simpler, well-scoped subtasks to small models while retaining a strong model for overall planning and synthesis (Fourney et al., 2024; Hadfield et al., 2025; Belcak et al., 2025; An-

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.

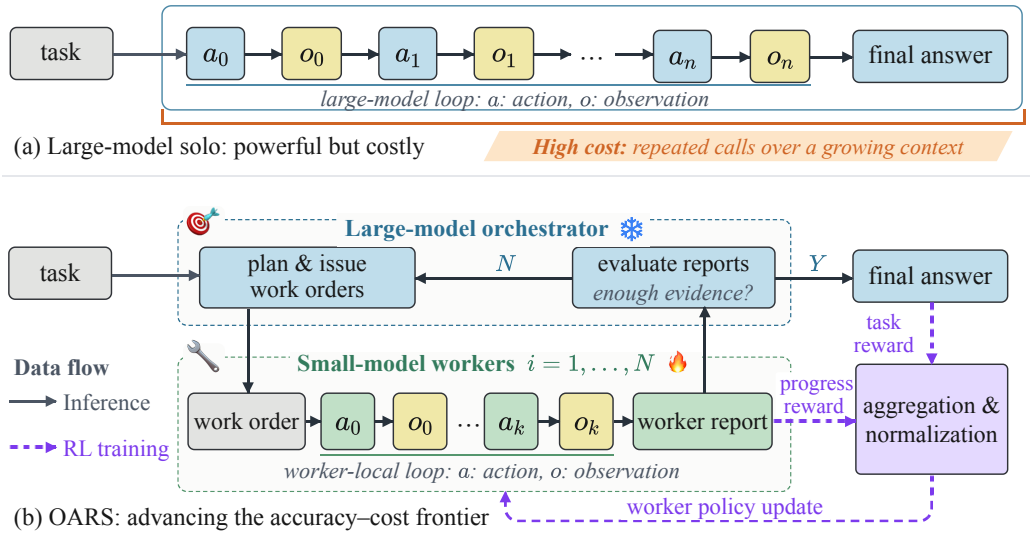


Figure 1: **OARS overview.** A unified work-order protocol separates global coordination from local execution: workers return only concise, task-relevant findings to the orchestrator rather than their full execution histories (Section 2). To effectively optimize small-model workers, OARS introduces contribution-aware rewards and hierarchical credit normalization during post-training (Section 3).

thropic, 2026a). However, cheaper workers alone do not guarantee system-level savings: Anthropic (2026a) reports that its multi-agent system still incurred 28–43% higher cost to achieve the same BrowseComp accuracy as a standalone large-model agent due to additional coordination overhead. Therefore, achieving consistent cost savings through multi-agent collaboration across complex and diverse tasks remains challenging. The best of both worlds—substantially lower costs with maintained or even improved task accuracy—has yet to be achieved through a unified, open approach.

To this end, we introduce OARS (*Orchestrator-Aware Role Specialization*), which advances the accuracy–cost frontier of agent systems (Figure 1). At the system level, OARS builds on a simple principle: effective coordination requires a shared interface, not shared execution contexts. A unified work-order protocol defines the boundary between global coordination by a large-model *orchestrator* and local execution by small *workers*, governing both information isolation and exchange. The orchestrator directs the task through this interface, receiving decision-relevant findings while workers retain their local execution histories. This division of labor itself enables stronger agent performance without introducing new models. With the same large model serving as both orchestrator and workers, the resulting multi-agent system improves accuracy by an average of 10.7 percentage points over single-agent execution (Table 1).

However, simply replacing these large-model workers with smaller models does not guarantee effective collaboration. To achieve competitive accuracy, small models must learn to fulfill their roles within the collaborative process. OARS therefore post-trains small workers in the complete multi-agent system. We identify a central challenge in reinforcement learning: the *free-rider problem*, in which a worker can be rewarded for collective success regardless of its own contribution, as a strong orchestrator or another worker may compensate for an unhelpful report, thereby masking the worker's failure to contribute. OARS addresses the *free-rider problem* through contribution-aware training that combines task-level rewards with worker-level progress rewards. By assessing whether each report advances the task, these progress rewards distinguish individual contributions from collective success. To balance credit assignment, we further introduce hierarchical credit normalization, preventing episodes with more workers or longer worker trajectories from receiving disproportionate training weight.

OARS substantially advances the accuracy–cost frontier of agentic systems. On BrowseComp-Plus (Chen et al., 2026c), OARS—with 8B workers assisting 230–550B-parameter orchestrators—improves accuracy over standalone large-model agents by 6.3 percentage points on average while being up to 10.6× more cost-efficient. We further introduce DR-256, an open benchmark enabling reproducible evaluation of deep-research agents without costly external search APIs. On DR-256,

OARS improves accuracy over standalone large-model agents by 19.9 percentage points on average while achieving up to a 4.2× reduction in cost per correct answer (Tables 1 and 2). OARS also extends to agentic coding, cutting cost per resolved task by 66.5% on SWE-bench Verified (Jimenez et al., 2024; OpenAI, 2024) while maintaining accuracy (Table 4).

2 OARS: A Unified Protocol for Multi-Agent Collaboration

We introduce OARS, which organizes model collaboration through a unified work-order protocol that divides complex tasks into local objectives. The central principle is that *effective coordination requires a shared interface, not shared execution contexts.* Workers may need extensive environment interaction, while the orchestrator needs only findings that guide its next decision. By delegating local action planning and observation processing to workers, OARS allows the orchestrator to coordinate the overall task while keeping workers' full execution histories outside its context (Figure 1).

For a task x in environment $\mathcal{E}$, OARS pairs an orchestrator $\mathcal{O}$ with workers sharing a policy π_θ. At round k, $\mathcal{O}$ uses its coordination history H_k to either return a final answer or issue work orders $b_{k,1}, \ldots, b_{k,m_k}$. Each order $b_{k,j}$ is executed independently by a worker through interactions with $\mathcal{E}$, yielding

$$(\tau_{k,j}, z_{k,j}) \sim \mathrm{Rollout}(\pi_\theta, b_{k,j}; \mathcal{E}), \tag{1}$$

where $\tau_{k,j}$ and $z_{k,j}$ denote the worker's local execution trajectory and returned report, respectively. Only workers interact directly with $\mathcal{E}$. The orders $b_{k,j}$ and reports $z_{k,j}$ enter H_{k+1}, while worker trajectories $\tau_{k,j}$ remain local. Reports from the same round are collected before the orchestrator's next decision. Both the number of work orders and the scope of each order can vary as the task evolves.

Work-order interface. The orchestrator generates work orders based on H_k through function calls with named arguments. The interface exposes three operations:

$$\begin{aligned}
&\texttt{spawn_worker}(\texttt{work_order} = b),\\
&\texttt{message_worker}(\texttt{worker_id} = i, \texttt{message} = m),\\
&\texttt{finish}(\texttt{answer} = y, \texttt{explanation} = e).
\end{aligned}$$

Here, b is a self-contained task brief, i identifies an existing worker, m is a follow-up instruction, and (y, e) is the final answer and its justification. The task brief is written in natural language, with the orchestrator instructed to specify the local objective, relevant context, and requirements for the returned evidence.

Before dispatching a work order, the harness parses the function name and arguments. Spawning a new worker requires a nonempty brief and an available slot under the pre-defined per-task worker cap; a follow-up requires an existing worker identifier. Unparseable calls, empty briefs, and unknown worker identifiers receive corrective feedback. Each accepted new order receives a worker identifier assigned by the OARS harness. Once the per-task worker cap is reached, the harness rejects additional spawn requests and instructs the orchestrator to reason over existing reports. Appendix D provides an illustrative example of the work-order protocol.

Worker execution. Each accepted work order is dispatched to a separate worker context containing the instructions and task brief, without exposing H_k or other workers' histories. At each turn, the worker may either interact with the environment through predefined interfaces or return a report to the orchestrator. These environment interfaces are unavailable to the orchestrator, and workers cannot communicate with or delegate to one another. Local execution is subject to turn and context limits; the harness returns corrective feedback for unsupported tool names or missing required arguments.

Reporting, follow-up, and finalization. A worker submits a report $z = (a, e)$ through its local `finish` call, where `answer` supplies a and `explanation` supplies e. The worker is instructed to include supporting evidence, such as source excerpts or test results. The harness returns each report z to the orchestrator together with the worker identifier, while the worker's full execution history remains local. Empty worker answers trigger a resubmission prompt. As workers approach

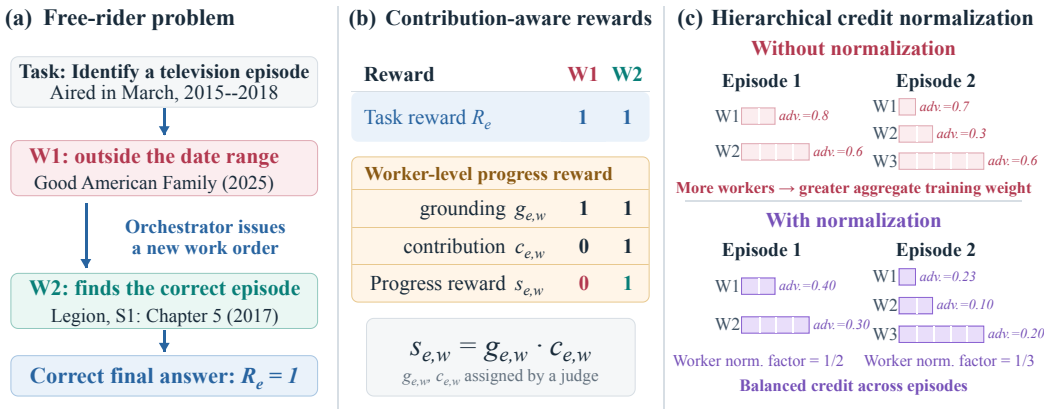


Figure 2: **OARS post-training.** (a) Collective success can conceal unhelpful work. (b) Contribution-aware rewards identify useful worker contributions. (c) Hierarchical credit normalization prevents episodes with more workers or longer trajectories from receiving disproportionate training weight. For clarity, panel (c) illustrates worker-count normalization, which reweights worker advantages; the same principle balances credit across turns within each worker trajectory.

the context limit, the harness reminds them to return their best available answer and supporting evidence.

After incorporating reports into H_{k+1}, the orchestrator refines the remaining work. It can issue new work orders to address unresolved questions or explore possibilities suggested by the reports. When further work builds on an earlier investigation, the orchestrator can send a follow-up instruction m to the corresponding worker i through `message_worker`, requesting clarification, additional evidence, or continued investigation. If no further work is needed, the orchestrator synthesizes the findings into a final answer y with justification e and concludes the task through `finish`.

3 OARS: Post-Training for the Accuracy–Efficiency Frontier

Effective collaboration depends on workers that can reliably carry out delegated tasks. While large models can often fulfill this role in a zero-shot setting, smaller models may struggle to provide accurate and useful information for subsequent decisions. Unhelpful reports can misdirect the orchestrator and compromise task success. We therefore post-train the shared worker policy to better fulfill this role, with a frozen large-model orchestrator providing global coordination.

3.1 The Free-Rider Problem

While task delegation lets each worker focus on a simpler subtask, the final outcome reflects the joint performance of the orchestrator and all workers, making it difficult to determine which behaviors should be reinforced. For example, on a BrowseComp question seeking a television episode aired in March 2015–2018, one worker returns a 2025 series. The orchestrator eventually obtains the correct answer from another worker after further verification. More generally, an unhelpful or even harmful report may be bypassed using evidence already available from other workers, or compensated for through additional rounds of delegation. In either case, the system succeeds despite the report, not because of it (Figure 2(a)).

Under outcome-based GRPO (Shao et al., 2024), however, all workers share the task-level reward regardless of their individual contributions. This creates a *free-rider problem*: workers that contribute little to task completion can still receive positive advantage from success achieved by their collaborators (Table 6). The resulting updates conflate useful execution with ineffective behavior masked by system-level compensation, dispersing credit beyond the behavior responsible for success and weakening the learning signal for worker specialization.

3.2 CONTRIBUTION-AWARE REWARDS

Rewarding workers solely for episode-level success would credit work that contributed little or nothing, whereas rewarding a worker only for providing the final answer would overlook useful intermediate findings. A worker need not solve the entire task to contribute. We instead assess whether each report provides information that advances the task. During training, a judge assigns a contribution score $c_{e,w} \in [0, 1]$ to worker w in episode e based on its work order, report, execution observations, and the reference answer. A grounding factor $g_{e,w} \in [0, 1]$ discounts claims unsupported by the worker's observations, so that an unsupported guess is not credited as highly as an evidence-backed finding. The worker-level progress reward is

$$s_{e,w} = g_{e,w} c_{e,w}. \tag{2}$$

The task reward $R_e \in \{0, 1\}$ indicates whether the collaborative episode succeeds; $s_{e,w}$ distinguishes the contributions of workers sharing that outcome. We retain both signals: outcome feedback aligns learning with the system's objective, while progress feedback makes credit sensitive to the information each worker provides. For episodes sampled for the same task, we standardize the two branches separately:

$$A_e^{\text{ep}} = \frac{R_e - \mu_R}{\sigma_R + \varepsilon}, \qquad A_{e,w}^{\text{wrk}} = \frac{s_{e,w} - \mu_s}{\sigma_s + \varepsilon}, \tag{3}$$

Here, μ_R and σ_R are the mean and standard deviation of task rewards across episodes in the group, with each episode counted once. μ_s and σ_s are the episode-balanced statistics of worker-level progress rewards defined in Section 3.3, and ε is a numerical stabilizer. Figure 2(b) illustrates how workers sharing the same task reward receive different progress rewards based on their grounding and contribution. Their training signal is therefore no longer determined solely by collective success.

3.3 HIERARCHICAL CREDIT NORMALIZATION

Contribution-aware rewards identify useful worker contributions, but their influence on training also depends on how samples are aggregated (Figure 2(c)). Episodes invoke varying numbers of workers, and workers generate trajectories spanning different numbers of interaction turns. Pooling all worker scores equally gives episodes with more workers greater weight in the reference statistics. Likewise, treating each interaction turn as an equally weighted training sample can give workers with longer trajectories greater influence on the training objective, regardless of their contribution. We therefore normalize both learning signals across episodes (inter-episode), while additionally normalizing contribution feedback across workers within each episode (intra-episode).

For a group of episodes sampled for the same task, we normalize worker-level progress rewards using episode-balanced statistics. Let $\mathcal{S}_e$ contain the workers with valid scores in episode e, $M_e = |\mathcal{S}_e|$, and $\mathcal{V} = \{e : M_e > 0\}$ within this group. We average within each episode before averaging across episodes:

$$\mu_s = \frac{1}{|\mathcal{V}|} \sum_{e \in \mathcal{V}} \frac{1}{M_e} \sum_{w \in \mathcal{S}_e} s_{e,w}, \qquad \sigma_s^2 = \frac{1}{|\mathcal{V}|} \sum_{e \in \mathcal{V}} \frac{1}{M_e} \sum_{w \in \mathcal{S}_e} (s_{e,w} - \mu_s)^2. \tag{4}$$

Each valid episode contributes equally to these statistics, regardless of its worker count, while individual score differences are retained in the variance. Let N_e be the total number of worker trajectories in episode e. We assign worker w the weighted advantage

$$A_{e,w} = \left[A_e^{\text{ep}} \frac{\bar{N}}{N_e} + A_{e,w}^{\text{wrk}} \frac{\bar{N}}{M_e} \right] / 2, \tag{5}$$

where $\bar{N}$ is the average number of worker trajectories per episode in the rollout batch. The episode's advantage is shared equally among its N_e workers, so invoking more workers does not amplify its training signal. Each scored worker's contribution advantage is similarly weighted by $1/M_e$. The same principle applies at the worker level: each worker's contribution advantage is normalized across the turns of its trajectory, so additional turns do not amplify its total credit.

Regarding the training objective, generated-token losses are averaged across all turns of each worker trajectory, preventing longer trajectories from receiving greater training weight. For a batch $\mathcal{B}$ of

worker trajectories, let $\mathcal{I}_{e,w}$ denote all generated-token positions in worker w's trajectory. The policy objective is

$$\mathcal{L}_{\text{policy}}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{(e,w)\in\mathcal{B}} \frac{1}{|\mathcal{I}_{e,w}|} \sum_{i\in\mathcal{I}_{e,w}} \ell_{\text{policy}}(\theta; i, A_{e,w}), \tag{6}$$

where ℓ_{policy} is the token-level clipped policy loss.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate OARS on tasks requiring sustained interaction with external environments, spanning web browsing and deep research across difficulty levels. For BrowseComp-Plus (Chen et al., 2026c), we use a 150-question evaluation split following Sun et al. (2025). We also introduce **DR-256** (Appendix C), an open, self-contained benchmark for multi-hop deep research. We construct a 256-question evaluation set through systematic curation of DeepResearch-9K (Wu et al., 2026), including evidence validation and filtering out questions solvable through single-step retrieval. We build a frozen web corpus and local search engine around these questions, enabling reproducible, off-the-shelf evaluation without external search APIs. MuSiQue (Trivedi et al., 2022) and FRAMES (Krishna et al., 2025) serve as supplementary benchmarks. Additionally, we assess OARS on SWE-bench Verified (Jimenez et al., 2024; OpenAI, 2024) for repository-level coding. Evaluation results are reported as averages over three runs.

Models. We select orchestrators spanning 230–550B total parameters, including MiniMax M2.7 (Chen et al., 2026a), DeepSeek V4 Flash (DeepSeek, 2026), and Nemotron Ultra (NVIDIA, 2026). We post-train Qwen3-8B workers (Yang et al., 2025) through supervised fine-tuning followed by reinforcement learning on the 680-question BrowseComp-Plus training set (Chen et al., 2026c; Sun et al., 2025) (disjoint from the evaluation set). For coding, we pair M2.7 with Qwen3.5-9B workers (Qwen Team, 2026). Further training details are provided in Appendix A.

Baselines. For each orchestrator, we compare OARS against three configurations: *Solo*, where the large model completes the task without workers; *Large workers*, where the same large model serves as both orchestrator and workers; and *Qwen3-8B workers*, which use the original checkpoint without post-training. We also compare OARS with AFM (Li et al., 2025b), MATPO (Mo et al., 2025), and RouteLLM (Ong et al., 2024) to assess its accuracy–cost trade-off against existing approaches.

4.2 Main Results

OARS advances the accuracy–cost frontier. Table 1 reports OARS accuracy across three orchestrators and two benchmarks. With 8B workers, OARS outperforms the large-model *Solo* baseline for all three models, improving accuracy by 3.6–9.6 percentage points on BrowseComp-Plus and 18.4–21.5 points on DR-256. It also achieves comparable or even higher accuracy than systems with large models acting as workers, yielding an average gain of 2.5 percentage points across the six model–benchmark combinations.

All these results are obtained with the same 8B worker post-trained on BrowseComp-Plus under M2.7. Without further fine-tuning, the worker collaborates effectively with DeepSeek V4 Flash and Nemotron Ultra, demonstrating transfer across orchestrators. Generalization also extends across benchmarks: our worker maintains its effectiveness on DR-256 despite having no exposure to the dataset during post-training. These gains underscore that the learned worker capabilities generalize beyond the specific orchestrator and benchmark used for post-training.

These accuracy gains come with substantially lower cost (Table 2). Across both benchmarks, OARS reduces cost per correct answer by 67.0–90.5% relative to the large-model *Solo* baseline, achieving higher accuracy with lower expense under every orchestrator. The advantage is more pronounced over systems with large models acting as workers: OARS achieves higher accuracy on average at only 3.8–10.1% of their cost. With trained small workers, OARS shifts both local execution and the processing of accumulated observations away from the large model, retaining it for global

Table 1: **Accuracy comparison between OARS and baselines.** Results are reported overall and by difficulty on BrowseComp-Plus and DR-256.

Large Model	Setup	BrowseComp-Plus Accuracy (%)				DR-256 Accuracy (%)		
		Overall	Easy	Medium	Hard	**Overall**	Hard	Challenge
—	Qwen3-8B solo	23.3	46.0	19.3	4.7	18.1	32.6	3.6
MiniMax M2.7	Solo	75.1	99.3	88.0	38.0	30.7	44.8	16.7
	Large workers	**80.0**	100.0	90.0	50.0	45.7	59.4	32.0
	Qwen3-8B workers	56.4	92.7	54.7	22.0	34.5	49.5	19.5
	OARS (ours)	78.7	100.0	84.7	51.3	**49.1**	63.5	34.6
DeepSeek V4 Flash	Solo	71.1	100.0	84.0	29.3	38.7	51.6	25.8
	Large workers	76.7	100.0	81.3	48.7	57.0	72.1	41.9
	Qwen3-8B workers	58.0	88.7	62.7	22.7	48.7	61.2	36.2
	OARS (ours)	**76.9**	100.0	81.3	49.3	**58.6**	75.3	41.9
Nemotron Ultra	Solo	67.3	96.0	70.0	36.0	37.0	51.0	22.9
	Large workers	73.1	98.0	77.3	44.0	51.4	69.3	33.6
	Qwen3-8B workers	59.6	91.3	64.0	23.3	50.7	65.6	35.7
	OARS (ours)	**76.9**	98.7	82.0	50.0	**58.5**	72.7	44.3

Table 2: **OARS advances the accuracy–cost frontier compared with baselines.**

Large Model	Setup	BrowseComp-Plus					DR-256				
		Acc. (%)	**Cost** (cent)	**Tokens (K)**			**Acc.** (%)	**Cost** (cent)	**Tokens (K)**		
				Prefill	Cache	Decode			Prefill	Cache	Decode
MiniMax M2.7	Solo	75.1	13.1	945	812	8.03	30.7	11.9	316	274	6.13
	Large workers	**80.0**	17.1	1,196	1,009	17.0	45.7	55.2	2,224	1,868	27.7
	OARS (ours)	78.7	**1.2**	32.3	27.8	5.60	**49.1**	**2.8**	53.2	47.3	7.68
DeepSeek V4 Flash	Solo	71.1	10.8	952	814	3.52	38.7	8.4	247	202	7.6
	Large workers	76.7	18.6	1,621	1,381	13.2	57.0	42.9	2,006	1,610	36.2
	OARS (ours)	**76.9**	**1.4**	54.2	49.9	6.13	**58.6**	**2.8**	21.4	17.3	10.8
Nemotron Ultra	Solo	67.3	21.2	980	904	6.57	37.0	16.7	333	291	5.25
	Large workers	73.1	41.6	1,518	1,272	24.7	51.4	108.9	3,494	3,114	26.7
	OARS (ours)	**76.9**	**4.2**	43.3	37.9	11.8	**58.5**	**4.1**	25.4	21.8	9.11

* Token usage is averaged per task and reported in thousands. Cache reports measured cache-hit tokens; cost is reported in US cents per correct answer.

coordination. For example, with M2.7 on BrowseComp-Plus, large-model prefill falls from 945K to 32.3K tokens per task, a 96.6% reduction, while accuracy rises from 75.1% to 78.7%. The resulting efficiency gain therefore accompanies stronger task performance, showing that trained small workers can preserve the benefits of collaboration while substantially reducing reliance on large-model computation. More details are provided in Appendix B.2.

Additional evaluations. We also benchmark OARS against two post-trained multi-agent baselines, AFM and MATPO (Li et al., 2025b; Mo et al., 2025). We further use RouteLLM (Ong et al., 2024) to route tasks between MATPO-14B and M2.7, forming the strongest baseline across all four benchmarks. Compared with this baseline, OARS

Table 3: **Comparison with existing systems.** Numbers are accuracy (%) across four benchmarks.

Method	BC-Plus	DR-256	MuSiQue	FRAMES
MATPO (14B)	41.3	37.5	43.7	36.7
AFM (7B)	22.7	23.4	29.7	15.3
AFM (32B)	44.7	44.9	54.7	38.0
RouteLLM + MATPO	74.7	45.1	65.3	72.7
OARS (ours)	**78.7**	**49.1**	**65.7**	**74.7**

improves accuracy on all four benchmarks while reducing cost per correct answer by 92.8%, 94.1%, 88.0%, and 89.4%, respectively.

OARS for Agentic Coding. On SWE-bench Verified, OARS pairs M2.7 with a Qwen3.5-9B worker post-trained on 350 tasks and is evaluated on a disjoint 150-task split. OARS resolves 60.4% of tasks compared with 61.3% for M2.7 alone, while reducing cost per resolved task by 66.5% (Table 4). These results demonstrate the effectiveness of OARS for cost-efficient repository-level coding.

Table 4: OARS on SWE-bench Verified. Cost is reported in US cents per resolved task.

System	**Accuracy** (%)	**Cost** (cent)
Qwen3.5-9B solo	34.4	—
M2.7 solo	61.3	5.59
M2.7 + small worker	53.6	2.19
M2.7 + OARS worker	60.4	1.87

4.3 Ablations and Analysis

Breakdown of Accuracy Gains. Figure 3 compares zero-shot, SFT, and RL-trained workers under the same M2.7 orchestrator. SFT raises overall accuracy from 56.4% to 73.1%, while subsequent RL reaches 78.7%. RL provides its largest gain on Hard questions, improving accuracy from 37.3% to 51.3% (+14.0 points), with modest gains on Easy and Medium questions.

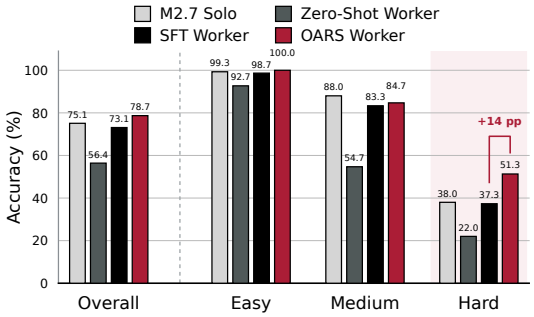
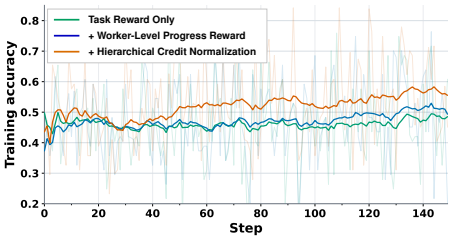


Figure 3: OARS workers undergo SFT followed by RL, improving accuracy on Hard questions by 14.0 percentage points over SFT alone.

Reward Design Effectiveness. Table 5 compares reward designs under raw and SFT initialization. Task-only GRPO yields limited gains, whereas adding worker-level progress rewards improves accuracy under both initializations. Hierarchical credit normalization sustains higher training accuracy and further improves evaluation accuracy by 2.9–3.6 percentage points.

Table 5: Reward design analysis. Left: evaluation accuracy across RL strategies and worker initializations. Right: rollout accuracy curves during RL training; faint lines show unsmoothed values and solid lines show TWEMA-smoothed values.

RL strategy \ Worker initialization	**Raw**	**SFT**
No RL	56.4	73.1
Task reward only	57.1	72.0
+ Worker-level progress reward	58.4	75.1
+ Hierarchical credit normalization	**61.3**	**78.7**



Contribution-Aware Rewards and Hierarchical Credit Normalization. Table 6 analyzes 2,620 worker calls from 450 RL episodes. To classify workers by their contributions, we rerun the orchestrator's final decision after removing one worker report at a time. In originally correct episodes, a worker is *pivotal* if removing its report changes the final answer from correct to incorrect. Otherwise, it is *constructive* if its report provides useful findings or verification, and a *free-rider* if it does not. These categories account for 8.5%, 35.3%, and 21.4% of all worker calls, respectively; the remaining 34.8% come from unsuccessful episodes.

Notably, task-only rewards assign positive advantage even to *free-riders*. Adding worker-level progress rewards makes their average advantage negative while preserving positive advantage for *pivotal* and *constructive* workers. Hierarchical credit normalization further increases the relative

Table 6: **Credit assignment under different reward and normalization schemes.** Values report mean advantage per worker call, with advantages scaled to unit variance within each scheme. Gini measures the concentration of credit across episodes; lower values indicate a more balanced allocation.

Reward signals	Credit normalization	Worker category				Gini↓
		Pivotal	Constructive	Free-rider	Wrong episode	
Task reward only	No	+0.02	+0.21	+0.50	-0.53	0.87
Task + progress	No	+0.23	+0.41	-0.33	-0.27	0.46
Task + progress	Yes	+0.33	+0.26	-0.29	-0.17	0.28

advantage of *pivotal* workers and distributes credit more evenly across episodes, reducing Gini from 0.46 to 0.28. Appendix B.1 shows that this pattern remains consistent across judge models.

5 Related Work

Multi-agent systems. Multi-agent systems enable LLMs to collaborate on complex tasks, often improving performance at additional cost (Li et al., 2023; Hong et al., 2023; Wu et al., 2023; Rajasekaran et al., 2025). ReWOO (Xu et al., 2023) reduces redundant token usage by separating tool-use planning from execution. Long-horizon agents also control context growth through history compression (Ye et al., 2026; Kang et al., 2025; Sun et al., 2025) and memory management (Zhou et al., 2026; Zhang et al., 2026). FrugalGPT (Chen et al., 2023) uses an answer-quality estimator to determine whether to invoke additional models, while RouteLLM (Ong et al., 2024) routes queries between stronger and weaker models. Minions (Narayan et al., 2025) introduces a collaboration protocol that offloads document reading from cloud models to smaller on-device models. Lyu et al. (2026); Su et al. (2025) also distribute task execution by handing off challenging parts to external models. OARS post-trains small-model workers to handle the majority of the workload, substantially advancing the accuracy–cost frontier on complex, long-horizon agentic tasks.

Reinforcement learning for LLM agents. Reinforcement learning has proven effective for improving LLM agents (Wang et al., 2025c; Xi et al., 2026a). Search-R1 (Jin et al., 2025) learns to interleave reasoning and search, with tool outputs masked from the policy loss. ReTool (Feng et al., 2025) trains LLMs to decide when to invoke code execution during reasoning. RL has also been applied to tasks such as multi-step web research (Wu et al., 2025; Li et al., 2025a; Zheng et al., 2025; Jiang et al., 2025) and software engineering (Wei et al., 2025b; Luo et al., 2025). In addition, MALT (Motwani et al., 2025) and MATPO (Mo et al., 2025) explore RL for agentic systems with distinct roles. ToolRL (Qian et al., 2025) further examines reward design by separately rewarding tool selection and argument correctness, with other work refining credit assignment at the action and agent levels (Wang et al., 2025a; Xi et al., 2026b; Li et al., 2026). OARS identifies free-riding as a central challenge in multi-agent RL and addresses it through contribution-aware training and hierarchical credit normalization.

6 Conclusion

We introduced OARS, which advances the accuracy–cost frontier of LLM agents by co-designing heterogeneous multi-agent collaboration and an effective post-training strategy. Our work-order protocol isolates small-model workers' local execution contexts from the large-model orchestrator, reducing its context growth and computational workload. Our post-training strategy, in turn, optimizes small workers for their collaborative roles, with contribution-aware training and hierarchical credit normalization addressing credit-assignment challenges in multi-agent reinforcement learning. Across BrowseComp-Plus and DR-256, OARS consistently improves accuracy over standalone large-model agents while being up to 10.6× and 4.2× more cost-efficient, respectively. On SWE-bench Verified, it cuts cost per resolved task by 66.5% while maintaining accuracy. We believe OARS will catalyze further research toward more capable and cost-efficient multi-agent systems.

AI USE STATEMENT

In this work, we used generative AI tools to assist with code development and improve the clarity and readability of the manuscript. Part of the training data (e.g., model responses used for reinforcement learning) was generated by LLMs as detailed in the paper. We have not used generative AI tools to formulate research ideas and methodology, implement the core methods, design experiments, interpret results, or formulate claims about system performance. The authors carefully reviewed and validated the AI-assisted work and checked all AI-edited text to ensure that it preserved the intended meaning. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

OARS makes capable LLM agents more accessible by substantially reducing their computational costs without sacrificing accuracy. Lower execution costs can broaden access to useful agent capabilities, while also making misuse easier to scale. Deployments should preserve applicable model safeguards and maintain human oversight for consequential actions. We also introduce DR-256, a new evaluation benchmark for deep-research agents. Its release will respect the licensing and privacy constraints of the underlying sources and document their provenance and conditions of use.

REFERENCES

Anthropic. Optimizing for Cost and Intelligence. Claude Platform Docs, 2026a. URL https://platform.claude.com/docs/en/about-claude/models/optimizing-for-cost-and-intelligence.

Anthropic. System card: Claude Opus 4.6. Anthropic, February 2026b. URL https://anthropic.com/claude-opus-4-6-system-card.

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. SWE-Search: Enhancing Software Agents with Monte Carlo Tree Search and Iterative Refinement. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.20285.

Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. Small Language Models are the Future of Agentic AI. *arXiv preprint arXiv:2506.02153*, 2025. URL https://arxiv.org/abs/2506.02153.

Aili Chen, Aonian Li, Baichuan Zhou, Bangwei Gong, Binyang Jiang, Boji Dan, et al. The MiniMax-M2 Series: Mini Activations Unleashing Max Real-World Intelligence. *arXiv preprint arXiv:2605.26494*, 2026a. URL https://arxiv.org/abs/2605.26494.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. *arXiv preprint arXiv:2305.05176*, 2023. URL https://arxiv.org/abs/2305.05176.

Tom Chen, Mao Cheng, Shi Dong, Kangrui Du, Yanbin Jiang, Jiajun Li, Yiming Li, Tao Lin, Yusheng Su, Andy Ye, Yueming Yuan, and Zhichen Zeng. Miles v0.1: Production-level post-training, 2026b. URL https://arxiv.org/abs/2609.08368.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Sahel Sharifymoghaddam, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Hosna Oyarhoseini, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. BrowseComp-Plus: A Fair and Disentangled Evaluation Benchmark for Deep Search Agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22349–22370. Association for Computational Linguistics, 2026c. doi: 10.18653/v1/2026.acl-long.1023. URL https://aclanthology.org/2026.acl-long.1023/.

DeepSeek. DeepSeek V4 Model Card. Official model card, 2026. URL https://fe-static.deepseek.com/chat/transparency/deepseek-V4-model-card-EN.pdf.

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

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. Magentic-One: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024. URL https://arxiv.org/abs/2411.04468.

Google DeepMind. Gemini 3.1 Pro Model Card, 2026a. URL https://deepmind.google/models/model-cards/gemini-3-1-pro/.

Google DeepMind. Gemini 3.6 Flash Model Card, 2026b. URL https://deepmind.google/models/model-cards/gemini-3-6-flash/.

Google DeepMind. Gemini 3.8 Flash Model Card, 2026c. URL https://deepmind.google/models/model-cards/gemini-3-8-flash/.

Jeremy Hadfield, Barry Zhang, Kenneth Lien, Florian Scholz, Jeremy Fox, and Daniel Ford. How We Built Our Multi-Agent Research System. Anthropic, 2025. URL https://www.anthropic.com/engineering/multi-agent-research-system.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023. URL https://arxiv.org/abs/2308.00352.

Thomas Hubert, Rishi Mehta, Laurent Sartran, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651(8106):607–613, 2026. doi: 10.1038/s41586-025-09833-y. URL https://www.nature.com/articles/s41586-025-09833-y.

Pengcheng Jiang, Xueqiang Xu, Jiacheng Lin, Jinfeng Xiao, Zifeng Wang, Jimeng Sun, and Jiawei Han. s3: You Don't Need That Much Data to Train a Search Agent via RL. *arXiv preprint arXiv:2505.14146*, 2025. URL https://arxiv.org/abs/2505.14146.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.06770.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Ö. Arık, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to Reason and Leverage Search Engines with Reinforcement Learning. In *Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2503.09516.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. ACON: Optimizing Context Compression for Long-horizon LLM Agents. *arXiv preprint arXiv:2510.00615*, 2025. URL https://arxiv.org/abs/2510.00615.

Satyapriya Krishna, Kalpesh Krishna, Anhad Mohananey, Steven Schwarcz, Adam Stambler, Shyam Upadhyay, and Manaal Faruqui. Fact, fetch, and reason: A unified evaluation of retrieval-augmented generation. In *Proceedings of NAACL*, 2025. URL https://arxiv.org/abs/2409.12941.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "Mind" exploration of large language model society. *arXiv preprint arXiv:2303.17760*, 2023. URL https://arxiv.org/abs/2303.17760.

Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, Weizhou Shen, Junkai Zhang, Dingchu Zhang, Xixi Wu, Yong Jiang, Ming Yan, Pengjun Xie, Fei Huang, and Jingren Zhou. WebSailor: Navigating Super-human Reasoning for Web Agent. *arXiv preprint arXiv:2507.02592*, 2025a. URL https://arxiv.org/abs/2507.02592.

Weizhen Li, Jianbo Lin, Zhuosong Jiang, Jingyi Cao, Xinpeng Liu, Jiayu Zhang, Zhenqiang Huang, Qianben Chen, Weichen Sun, Qiexiang Wang, Hongxuan Lu, Tianrui Qin, Chenghao Zhu, Yi Yao, Shuying Fan, Xiaowan Li, Tiannan Wang, Pai Liu, King Zhu, He Zhu, Dingfeng Shi, Piaohong Wang, Yeyi Guan, Xiangru Tang, Minghao Liu, Yuchen Eleanor Jiang, Jian Yang, Jiaheng Liu, Ge Zhang, and Wangchunshu Zhou. Chain-of-Agents: End-to-End Agent Foundation Models via Multi-Agent Distillation and Agentic RL. *arXiv preprint arXiv:2508.13167*, 2025b. URL https://arxiv.org/abs/2508.13167.

Yanming Li, Xuelin Zhang, Wenjie Lu, Ziye Tang, Maodong Wu, Haotian Luo, Tongtong Wu, Zijie Peng, Hongze Mi, Yibo Feng, Naiqiang Tan, Chao Huang, Lian Peng, and Li Shen. Who Deserves the Reward? SHARP: Shapley Credit-based Optimization for Multi-Agent System. *arXiv preprint arXiv:2602.08335*, 2026. URL https://arxiv.org/abs/2602.08335.

Michael Luo, Naman Jain, Jaskirat Singh, Sijun Tan, Ameen Patel, Qingyang Wu, Alpay Ariyak, Colin Cai, Tarun Venkat, Shang Zhu, Ben Athiwaratkun, Manan Roongta, Ce Zhang, Li Erran Li, Raluca Ada Popa, Koushik Sen, and Ion Stoica. DeepSWE: Training a Fully Open-sourced, State-of-the-Art Coding Agent by Scaling RL. Together AI, 2025. URL https://www.together.ai/blog/deepswe.

Niqi Lyu, Pengtao Shi, Wei Qiu, Jianlin Zhong, Sicong Xia, Jianyao Ma, and Yicheng Ding. PyroDash: Cost-Efficient Token-Level Small-Large Language Model Collaborative Inference. *arXiv preprint arXiv:2607.20327*, 2026. URL https://arxiv.org/abs/2607.20327.

Zhanfeng Mo, Xingxuan Li, Yuntao Chen, and Lidong Bing. Multi-agent tool-integrated policy optimization. *arXiv preprint arXiv:2510.04678*, 2025. URL https://arxiv.org/abs/2510.04678.

Moonshot AI. Kimi K2.6: Advancing open-source coding. Moonshot AI technical blog, 2026. URL https://www.kimi.ai/blog/kimi-k2-6.

Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev, Philip H. S. Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. MALT: Improving Reasoning with Multi-Agent LLM Training. In *Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2412.01928.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. WebGPT: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021. URL https://arxiv.org/abs/2112.09332.

Avanika Narayan, Dan Biderman, Sabri Eyuboglu, Avner May, Scott Linderman, James Zou, and Christopher Ré. Minions: Cost-efficient collaboration between on-device and cloud language models. *arXiv preprint arXiv:2502.15964*, 2025. URL https://arxiv.org/abs/2502.15964.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025. URL https://arxiv.org/abs/2506.13131.

NVIDIA. NVIDIA Nemotron 3 Ultra, 2026. URL https://research.nvidia.com/labs/nemotron/Nemotron-3-Ultra/.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs with preference data. *arXiv preprint arXiv:2406.18665*, 2024. URL https://arxiv.org/abs/2406.18665.

OpenAI. Introducing SWE-bench Verified, 2024. URL https://openai.com/index/introducing-swe-bench-verified/.

OpenAI. Introducing deep research. OpenAI, 2025. URL https://openai.com/index/introducing-deep-research/.

OpenAI. On the Navier–Stokes millennium prize problem. OpenAI Research, September 2026. URL https://openai.com/index/navier-stokes-solution/.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. ToolRL: Reward is All Tool Learning Needs. *arXiv preprint arXiv:2504.13958*, 2025. URL https://arxiv.org/abs/2504.13958.

Qwen Team. Qwen3.5-9B. Hugging Face model card, 2026. URL https://huggingface.co/Qwen/Qwen3.5-9B.

Prithvi Rajasekaran, Ethan Dixon, Carly Ryan, and Jeremy Hadfield. Effective Context Engineering for AI Agents. Anthropic, 2025. URL https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents.

Stephen E. Robertson, Steve Walker, Susan Jones, Micheline M. Hancock-Beaulieu, and Mike Gatford. Okapi at TREC-3. In *Proceedings of the Third Text REtrieval Conference (TREC-3)*, pp. 109–126. National Institute of Standards and Technology, 1995. URL https://www.microsoft.com/en-us/research/publication/okapi-at-trec-3/.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, Hongxu Yin, Yi Dong, Evelina Bakhturina, Tao Yu, Yejin Choi, Jan Kautz, and Pavlo Molchanov. ToolOrchestra: Elevating intelligence via efficient model and tool orchestration. *arXiv preprint arXiv:2511.21689*, 2025. URL https://arxiv.org/abs/2511.21689.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon LLM agent via context-folding. *arXiv preprint arXiv:2510.11967*, 2025. URL https://arxiv.org/abs/2510.11967.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 2022. URL https://arxiv.org/abs/2108.00573.

Hanlin Wang, Chak Tou Leong, Jiashuo Wang, Jian Wang, and Wenjie Li. SPA-RL: Reinforcing LLM Agents via Stepwise Progress Attribution. *arXiv preprint arXiv:2505.20732*, 2025a. URL https://arxiv.org/abs/2505.20732.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, et al. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. In *International Conference on Learning Representations*, 2025b. URL https://arxiv.org/abs/2407.16741.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. RAGEN: Understanding Self-Evolution in LLM Agents via Multi-Turn Reinforcement Learning. *arXiv preprint arXiv:2504.20073*, 2025c. URL https://arxiv.org/abs/2504.20073v2.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. BrowseComp: A Simple Yet Challenging Benchmark for Browsing Agents. *arXiv preprint arXiv:2504.12516*, 2025a. URL https://arxiv.org/abs/2504.12516.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. SWE-RL: Advancing LLM Reasoning via Reinforcement Learning on Open Software Evolution. In *Advances in Neural Information Processing Systems*, 2025b. URL https://arxiv.org/abs/2502.18449.

Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhenglin Wang, Zhengwei Tao, Ding-Chu Zhang, Zekun Xi, Robert Tang, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. WebDancer: Towards Autonomous Information Seeking Agency. In *Advances in Neural Information Processing Systems*, volume 38, pp. 120957–120985, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/af043aee9cb137785d93195c6cf4cd96-Abstract-Conference.html.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. *arXiv preprint arXiv:2308.08155*, 2023. URL https://arxiv.org/abs/2308.08155.

Tongzhou Wu, Yuhao Wang, Xinyu Ma, Xiuqiang He, Shuaiqiang Wang, Dawei Yin, and Xiangyu Zhao. DeepResearch-9K: A Challenging Benchmark Dataset of Deep-Research Agent. *arXiv preprint arXiv:2603.01152*, 2026. URL https://arxiv.org/abs/2603.01152.

Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Jiaqi Liu, Honglin Guo, Yajie Yang, Rui Zheng, Junjie Ye, Jiazheng Zhang, Wenxiang Chen, Wei He, Yiwen Ding, Guanyu Li, Zehui Chen, Zhengyin Du, Xuesong Yao, Yufei Xu, Jiecao Chen, Tao Gui, Zuxuan Wu, Qi Zhang, Xuanjing Huang, and Yu-Gang Jiang. AgentGym-RL: An Open-Source Framework to Train LLM Agents for Long-Horizon Decision Making via Multi-Turn RL. In *International Conference on Learning Representations (ICLR)*, 2026a. URL https://openreview.net/forum?id=ZgCCDwcGwn.

Zhiheng Xi, Chenyang Liao, Guanyu Li, Zhihao Zhang, Wenxiang Chen, Binghai Wang, Senjie Jin, Yuhao Zhou, Jian Guan, Wei Wu, Tao Ji, Tao Gui, Qi Zhang, and Xuanjing Huang. AgentPRM: Process Reward Models for LLM Agents via Step-Wise Promise and Progress. In *Proceedings of the ACM Web Conference 2026*, pp. 4184–4195. Association for Computing Machinery, 2026b. doi: 10.1145/3774904.3792551. URL https://doi.org/10.1145/3774904.3792551.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. ReWOO: Decoupling Reasoning from Observations for Efficient Augmented Language Models. *arXiv preprint arXiv:2305.18323*, 2023. URL https://arxiv.org/abs/2305.18323.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. *arXiv preprint arXiv:2405.15793*, 2024. URL https://arxiv.org/abs/2405.15793.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, Pengjun Xie, Fei Huang, Jingren Zhou, Siheng Chen, and Yong Jiang. AgentFold: Long-Horizon Web Agents with Proactive Context Folding. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=IuZoTgsUws.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic Context Engineering: Evolving Contexts for Self-Improving

Language Models. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.04618.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. DeepResearcher: Scaling Deep Research via Reinforcement Learning in Real-world Environments. *arXiv preprint arXiv:2504.03160*, 2025. URL https://arxiv.org/abs/2504.03160.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Bryan Kian Hsiang Low, and Paul Liang. MEM1: Learning to Synergize Memory and Reasoning for Efficient Long-Horizon Agents. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=XY8AaxDSLb.

A Training Details

Following Section 3, we post-train small-model workers for OARS. The worker policy is first initialized through supervised fine-tuning on trajectories generated by a larger teacher model acting as a worker under the same protocol. We retain trajectories from successful episodes and discard samples exceeding the worker's context window. The training loss is applied only to worker-generated tokens, with work orders and tool observations provided as context. OARS then refines the worker policy through contribution-aware reinforcement learning in the complete multi-agent system, while keeping the large-model orchestrator frozen. For coding tasks, we likewise post-train Qwen3.5-9B workers but omit a separate SFT initialization.

We conduct all post-training with M2.7 as the orchestrator and use the same model as the teacher worker for SFT trajectory collection. For reinforcement learning, we train the worker policy for 150 steps. Each step samples 16 question groups, with each group comprising 8 rollouts for the same question. Gemini-3.6-Flash (Google DeepMind, 2026b) serves as the worker judge for progress rewards. We implement asynchronous reinforcement learning with Miles (Chen et al., 2026b), allowing a maximum weight staleness of four steps. We use Adam with a constant learning rate of 1×10^{-6}, $(\beta_1, \beta_2) = (0.9, 0.98)$, a weight decay of 0.01, and gradient clipping at 1.0. Rollouts use a temperature of 1.0 and a maximum of 4,096 tokens per worker turn; worker trajectories are limited to 24,576 tokens for training. We optimize a clipped group-relative policy objective with lower and upper clipping coefficients of 0.2 and 0.28, respectively, together with a KL penalty of 0.001.

B Additional Experimental Results

B.1 Robustness to Judge Models

Table 7 examines the sensitivity of credit assignment to the choice of judge model. We rescore the same training traces with five different judge models (Google DeepMind, 2026a;b;c; DeepSeek, 2026). Although credit values vary across judges, all five assign positive average advantages to pivotal and constructive workers and negative advantages to free-riders. The distinction between useful and unhelpful work therefore remains consistent despite differences in individual judgments. Hierarchical credit normalization also reduces the Gini coefficient for every judge, from 0.46–0.48 to 0.28–0.32, consistently producing a more balanced allocation of credit across episodes. Together, these results demonstrate the robustness of contribution-aware rewards and hierarchical credit normalization to the choice of judge model.

Table 7: **Credit assignment across various judge models.** Values report mean advantage per worker call, with advantages scaled to unit variance within each judge model. Gini is reported before and after hierarchical credit normalization.

Judge Model	**Worker category**				**Gini↓**
	Pivotal	Constructive	Free-rider	Wrong episode	
Gemini-3.1-Pro	+0.36	+0.26	-0.30	-0.17	0.46→0.29
Gemini-3.6-Flash	+0.33	+0.26	-0.29	-0.17	0.46→0.28
Gemini-3.8-Flash	+0.35	+0.27	-0.30	-0.17	0.47→0.31
DeepSeek-V4-Pro	+0.29	+0.31	-0.31	-0.18	0.47→0.32
DeepSeek-V4-Flash	+0.38	+0.27	-0.31	-0.18	0.48→0.31

B.2 Detailed Token and Cost Analysis

Table 8 breaks down token usage and inference cost for OARS with Qwen3-8B and Qwen3-4B workers. Token counts are per-task averages in thousands; Cache reports measured cache-hit input tokens. Costs are reported in US cents per correct answer. For the orchestrators, we use public API rates at evaluation time, reported as uncached input/cached input/output in US dollars per million tokens: \$0.30/\$0.06/\$1.20 for MiniMax M2.7, \$0.44/\$0.014/\$1.32 for DeepSeek V4 Flash, and

Table 8: **Accuracy, token usage, and cost of OARS across worker sizes.**

Large Model	Setup	Acc. (%)	Cost (cent)	Orchestrator tokens (K)			Worker tokens (K)		
				Prefill	Cache	Decode	Prefill	Cache	Decode
BrowseComp-Plus									
MiniMax M2.7	Solo	75.1	13.1	945	812	8.03	—	—	—
	Large workers	**80.0**	17.1	21.5	17.9	4.54	1,174	991	12.5
	OARS (8B)	78.7	**1.2** (4.2)	32.3	27.8	5.60	511	372	13.5
	OARS (4B)	75.8	1.3 (**3.0**)	33.0	28.2	5.62	554	407	16.9
DeepSeek V4 Flash	Solo	71.1	10.8	952	814	3.52	—	—	—
	Large workers	76.7	18.6	40.8	36.7	4.77	1,580	1,344	8.47
	OARS (8B)	**76.9**	**1.4** (5.0)	54.2	49.9	6.13	603	439	15.7
	OARS (4B)	72.7	1.5 (**3.4**)	53.7	48.4	5.92	598	441	18.3
Nemotron Ultra	Solo	67.3	21.2	980	904	6.57	—	—	—
	Large workers	73.1	41.6	31.3	26.6	9.27	1,487	1,246	15.4
	OARS (8B)	**76.9**	**4.2** (7.6)	43.3	37.9	11.8	557	404	14.0
	OARS (4B)	76.0	4.5 (**6.2**)	42.6	36.3	12.5	569	417	16.8
DR-256									
MiniMax M2.7	Solo	30.7	11.9	316	274	6.13	—	—	—
	Large workers	45.7	55.2	47.8	41.1	8.42	2,176	1,827	19.3
	OARS (8B)	**49.1**	**2.8** (10.2)	53.2	47.3	7.68	931	744	17.0
	OARS (4B)	45.6	3.4 (**8.0**)	55.8	49.1	8.90	1,133	924	15.9
DeepSeek V4 Flash	Solo	38.7	8.4	247	202	7.60	—	—	—
	Large workers	57.0	42.9	17.3	13.0	12.7	1,989	1,597	23.5
	OARS (8B)	58.6	**2.8** (8.3)	21.4	17.3	10.8	823	655	15.1
	OARS (4B)	**59.2**	3.0 (**6.0**)	23.0	18.4	11.7	980	801	14.1
Nemotron Ultra	Solo	37.0	16.7	333	291	5.25	—	—	—
	Large workers	51.4	108.9	20.6	18.2	9.07	3,473	3,096	17.6
	OARS (8B)	58.5	**4.1** (9.5)	25.4	21.8	9.11	805	638	14.4
	OARS (4B)	**59.9**	4.2 (**7.3**)	27.7	23.8	9.42	1,010	825	13.7

Cost reports large-model API cost; parenthetical values include estimated OARS worker cost. Best values under each accounting are bolded.

$0.50/$0.10/$2.20 for Nemotron Ultra.[1] Both our worker models are small enough to be easily deployed locally without additional API costs. Even when we account for worker inference using token rates of $0.10/$0.02/$0.16 and $0.05/$0.01/$0.08 for the 8B and 4B workers, respectively, OARS still retains a substantial accuracy–cost advantage over the baselines.

C DR-256: An Open, Self-Contained Benchmark for Deep Research

Progress in deep-research agents requires evaluation environments that are both accessible and reproducible. However, reliance on online search APIs makes repeated experimentation costly, while service disruptions and temporal variability in search results further undermine reliable comparisons. We introduce DR-256 to lower these barriers: an open benchmark that brings multi-hop research tasks and a self-contained search environment into a single, reusable evaluation suite. Its frozen web corpus and local search engine support off-the-shelf evaluation without paid search APIs, allowing researchers to compare agents against a common evidence base.

[1] MiniMax API pricing; DeepSeek API pricing; Nemotron Ultra pricing via OpenRouter/DeepInfra. Accessed September 2026.

From research traces to an interactive benchmark. Building a reusable deep-research benchmark requires more than questions and reference answers: agents must be able to search for and inspect the evidence needed to solve them. We use DeepResearch-9K (Wu et al., 2026) as a starting point, drawing on its questions, reference answers, and retrieval traces to construct this environment. The traces record search queries and visited URLs but omit the retrieved content, providing a map of the evidence sources rather than an executable evaluation environment. We consolidate repeated questions and reconcile their retrieval provenance, identifying 41,913 canonical URLs associated with 6,646 questions. These sources form the basis of the searchable corpus described below.

Self-contained search environment. A controlled search environment should require agents to discover relevant evidence, not merely expose the documents containing the answer directly. We therefore retain the full collected corpus, including distractors, so that agents must identify relevant sources through retrieval and inspection. Source pages are converted to text and deduplicated, yielding 30,318 unique documents. We divide them into 800-word chunks with 100-word overlap, producing 243,615 chunks, and use BM25 ranking (Robertson et al., 1995) implemented in SQLite FTS5. Search returns the best-matching chunk for each URL; a separate document-opening interface provides access to the source text. The frozen corpus and local retrieval engine provide a consistent evidence environment across evaluations without external API services.

Validity, answerability, and difficulty. First, to prevent direct answer leakage, we exclude 28 documents containing benchmark questions and reference answers from the retrieval corpus. This prevents agents from bypassing evidence gathering through direct lookup. Second, to ensure that questions remain answerable from independent evidence, each retained question must have an independent source document containing its reference answer; these supporting documents are identified internally for validation, not supplied to agents as hints. Finally, we screen out questions answerable from search snippets, early page openings, or a single straightforward retrieval step. This yields 672 candidate questions for subsequent model-based difficulty calibration.

Model-based difficulty stratification. We evaluate each candidate question with three independent rollouts of an off-the-shelf Qwen3-8B model. Retained questions must have three complete, nonempty, successfully graded rollouts and require at least three interaction turns on average. The *Hard* group comprises questions solved in one or two rollouts. The *Challenge* group consists of questions that the model fails to solve across all three attempts. Questions answered correctly in every rollout are excluded. We randomly sample 128 questions from each group to form the final 256-question benchmark.

D Work Orders in Practice

The following example illustrates how an orchestrator delegates tasks and uses worker reports through the OARS work-order protocol. The original question and work orders are summarized for brevity, and reports are excerpted with omissions marked by ellipses.

Question (summarized). Identify a biscuit company from historical clues, then find the amount of butter in an early-2020s recipe for one of its products. The product originated around the turn of the twentieth century.

Orchestrator: reads the question and separates company identification from recipe retrieval, issuing the first order below.

Orchestrator → W1: spawn_worker(work_order=b_1)

```
"task":
  "objective": Identify the company described by the historical clues below.
  "context": Produces savory, sweet, plain, chocolate-filled, and flavored cookies; founded in the 1850s; premises burnt in the 1910s, followed by government protection; mascot introduced on packaging in the early 1920s; expanded into breakfast products in the 2010s.
  "requirements": Return the company name with evidence. Do not guess; rely on historical records, news articles, and corporate histories.
```

W1 interacts with the environment: five searches and one page opened, followed by the report below.

Report $z_1 = (a_1, e_1)$: W1 → orchestrator

"answer": Bakers Limited (commonly known as Bakers, also referred to as Baumann's Biscuits)
"explanation": Based on my research, the company described in the brief is Bakers Limited, commonly known as Bakers (originally Baumann's Biscuits). [...] In 1915, "the shop and bread and cake premises were burnt down." The factory survived the initial destruction and was "put under government protection to prevent further damage." [docid 75684] [...]

Orchestrator: extracts the company identity from W1's report and incorporates it into a targeted order for the recipe measurement.

Orchestrator → W2: spawn_worker(work_order=b_2)

"task":
"objective": Find the butter measurement in a recipe for a Bakers Limited product.
"context": Bakers Limited is a South African company. Locate an article from 2020–2023 about a product originating around 1900; candidates include Tennis biscuits, Romany Creams, Red Label, and Blue Label.
"requirements": The required butter measurement from the matching recipe.

W2 interacts with the environment: six searches and two pages opened, followed by the report below.

Report $z_2 = (a_2, e_2)$: W2 → orchestrator

"answer": 90 g
"explanation": [...] "Delicious homemade tennis biscuits: A taste of South African history!" published on 2023-09-08 on finglobal.com (FinGlobal). [...] The article explicitly states "Tennis biscuits were first manufactured around the turn of the twentieth century" [docid 80624]. [...] The article's recipe shows "Salted Butter, melted – 90 g" [docid 80624].

Orchestrator: combines the company identification with the recipe evidence and returns the final answer: **90 grams**. Only worker reports are returned to the orchestrator; full source documents and execution histories remain local to workers.