Learning to Orchestrate Agents in Natural Language with the Conductor

Nielsen et al., ICLR 2026

Presenter: *Haeyoung Lee*

June 05 , 2026

Seoul National University

Motivation

- Modern LLMs are powerful, but each model has different strengths across coding, math, scientific reasoning, and formatting.
- No single LM is universally optimal across all tasks.
- Existing agentic workflows such as routing, self-reflection, verification, and debate are useful, but usually hand-designed and task-specific.
- **Key question:** Can we train a small language model to automatically coordinate stronger worker LLMs?
- **RL Conductor** learns to *divide challenging problems*, *assign focused subtasks to worker LLMs*, and *design agent communication structures* in natural language.

Main Idea: The RL Conductor Framework

- The paper introduces a **Conductor model** trained with reinforcement learning to discover coordination strategies over a pool of worker LLMs.
- For each input question, the CONDUCTOR outputs a complete **agentic workflow** rather than directly producing the final answer.
- Each workflow step specifies three decisions:

Decision	Meaning
`model_id`	Which worker LLM should be called at this step.
`subtasks`	What natural-language instruction should be given to that worker.
`access_list`	Which previous worker responses should be visible in the current worker's context.

- This formulation allows the CONDUCTOR to construct flexible workflows such as routing, sequential reasoning chains, independent attempts, verification, refinement, and aggregation.
- The central idea is to learn **worker selection, focused subtask design, and information flow** jointly through end-to-end reward maximization.

Conductor Prompt

```
Your role is to obtain answers by iteratively querying powerful language
models, each with a different skillset.

Given a user question and a list of numbered language models, output a
sequence of up to 5 workflow steps.

Each workflow step contains three elements:
  1.  a language model to call,
  2.  a natural-language subtask for that model,
  3.  an access list specifying which previous steps it can see.

The final model's response will be returned as the final answer.

Your response should be provided as three Python lists:
  model_id = [...]
  subtasks = [...]
  access_list = [...]
```

- The prompt constrains the output to be parseable as a complete coordination strategy.
- Worker LLMs are represented as numbered models, encouraging the CONDUCTOR to learn model usage through reward feedback.
- Few-shot examples are included to condition the model toward valid orchestration formats.

Conductor Output

User

We call a subarray of an array complete if the number of distinct elements in the subarray is equal to the number of distinct elements in the whole array. Return the number of complete subarrays.

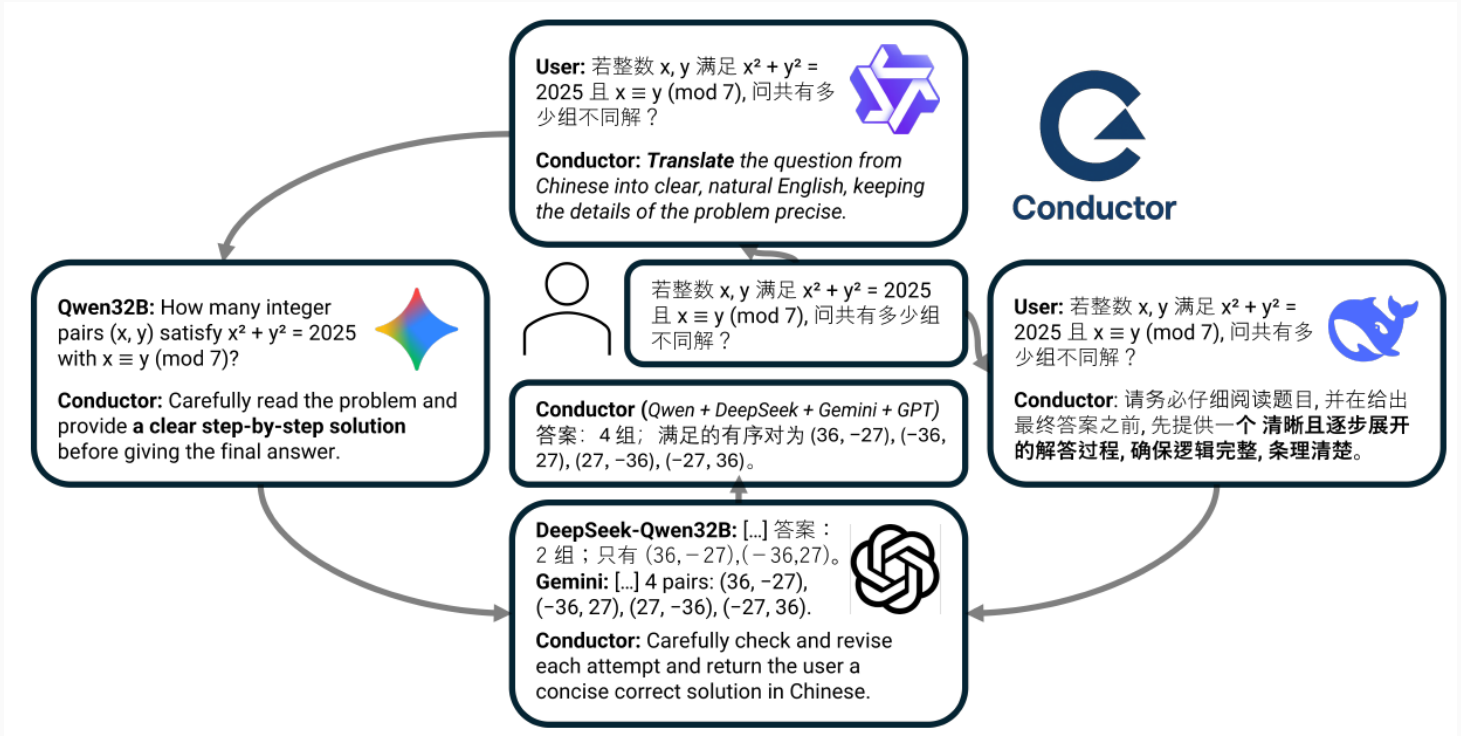
Conductor

Here's the plan: 1) Model 2 will develop an algorithm to efficiently count all possible complete subarrays for a given array, 2) Model 0 will implement the function to solve the problem.

model_id = [2, 0]
subtasks = ["Develop an efficient algorithm to count the number of complete subarrays of an array", "Implement the algorithm described by the previous agent in Python"]
access_list = [[] , ["all"]]

- `model_id`: the ordered sequence of worker LLMs to call.
- `subtasks`: the natural-language instructions assigned to each worker.
- `access_list`: whether each worker sees no previous response, selected responses, or all previous responses.

Example: Orchestrating Different Worker Skills



- The workflow combines translation, mathematical reasoning, independent solving, and final checking.
- Different workers are not simply averaged; they are assigned different roles in a structured workflow.
- The example illustrates how natural language can serve as a flexible interface for model coordination.

Training: GRPO (Group Relative Policy Optimization)

- **GRPO idea:** for the same question, sample multiple workflows and compare them within the group; workflows that perform better than the group average are reinforced.
- Training dataset:

$$\mathcal{D} = \{(q_i, s_i)\}_{i=1}^{N},$$

  where q_i is a verifiable question and s_i is the ground-truth solution.
- Conductor policy:

$$\pi_\theta(o \mid q_i),$$

  where π_θ is the trainable CONDUCTOR LLM, and o is a complete workflow output containing `model_id`, `subtasks`, and `access_list`.
- For each question q_i, GRPO samples a group of G candidate workflows:

$$\{o_1, \ldots, o_G\} \sim \pi_\theta(\cdot \mid q_i).$$

Training: Reward and Advantage

- Each sampled workflow o_j is executed by worker LLMs.
- The final worker response is checked against the ground-truth solution s_i.

$$r_j = \begin{cases} 0, & \text{if workflow } o_j \text{ cannot be parsed,} \\ 0.5, & \text{if } o_j \text{ is parseable but the final answer is incorrect,} \\ 1, & \text{if the final answer matches } s_i. \end{cases}$$

$$A_j = \frac{r_j - \mathrm{mean}(\{r_1, \ldots, r_G\})}{\mathrm{std}(\{r_1, \ldots, r_G\})} \tag{1}$$

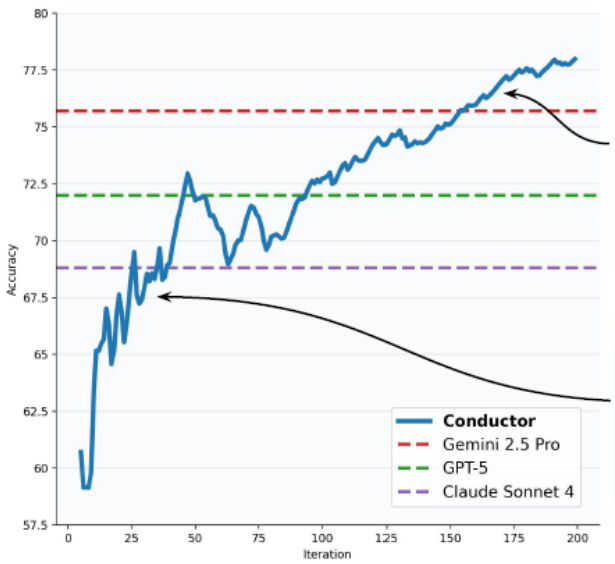
- r_j: reward assigned after the entire coordination strategy is executed.
- A_j: Monte-Carlo advantage computed by comparing r_j with other sampled workflows for the same question.
- If $A_j > 0$, workflow o_j performed better than the group average.

Training: GRPO Objective

$$J(\theta) = \mathbb{E}_{q_i \sim \mathcal{D},\, \{o_j\}_{j=1}^{G} \sim \pi_\theta(\cdot|q_i)} \Bigg[\frac{1}{G} \sum_{j=1}^{G} \Big(\min\left(r_j A_j,\ \mathrm{clip}(r_j, 1-\epsilon, 1+\epsilon) A_j\right) - \beta D_{\mathrm{KL}}(\pi_\theta \| \pi_{\mathrm{ref}}) \Big) \Bigg] \tag{2}$$

- $J(\theta)$: objective used to update the CONDUCTOR policy π_θ.
- The term $r_j A_j$ gives larger contribution to workflows with high reward and positive advantage.
- The clipping term bounds the reward-weighted contribution.
- $D_{\mathrm{KL}}(\pi_\theta \| \pi_{\mathrm{ref}})$ penalizes deviation from the reference model.
- β: KL penalty weight; in the main training setup, the authors set this penalty to 0.

What Does RL Learn?



...

subtasks = ["Understand the problem and find a strategy to calculate the punishment number. Identify the integers whose squares can be partitioned into substrings that sum to the integer itself. Then, calculate the sum of the squares of these integers.",

"Verify the proposed strategy and find any gaps or errors in the understanding. Provide a step-by-step verification process in <idea>tags.",

"Refine the proposed strategy if necessary and then implement the punishmentNumber function in Python according to the given constraints. Provide a step-by-step solution in <idea>tags and return the final code in <answer>tags as specified."]

...

subtasks = ["Analyze the given parameters of the simple harmonic motion (period and amplitude) and the initial position of the object. Identify the mathematical equation that describes the position of the object as a function of time.",

"Calculate the time it takes for the object to go from x = 5.70 cm to x = -1.60 cm using the equation derived in the previous step and the given parameters."]

- Early in training, the CONDUCTOR produces reasonable subtasks, but collaboration remains limited.
- As training progresses, planner roles, targeted instructions, shared intermediate reasoning, verification, and refinement emerge.
- The learned behavior reflects coordinated use of worker capabilities, rather than only selecting a single best model.

Datasets and Evaluation

- **Conductor:** Qwen2.5-7B, trained to generate workflows of up to 5 steps.
- **Worker pool:** GPT-5, Gemini-2.5-Pro, Claude-Sonnet-4, DeepSeek-R1-Distill-Qwen-32B, Gemma3-27B-it, Qwen3-32B, and Qwen3-32B thinking mode.
- Training uses 960 problems from MATH, MMLU, RLPR, and LiveCodeBench V1.

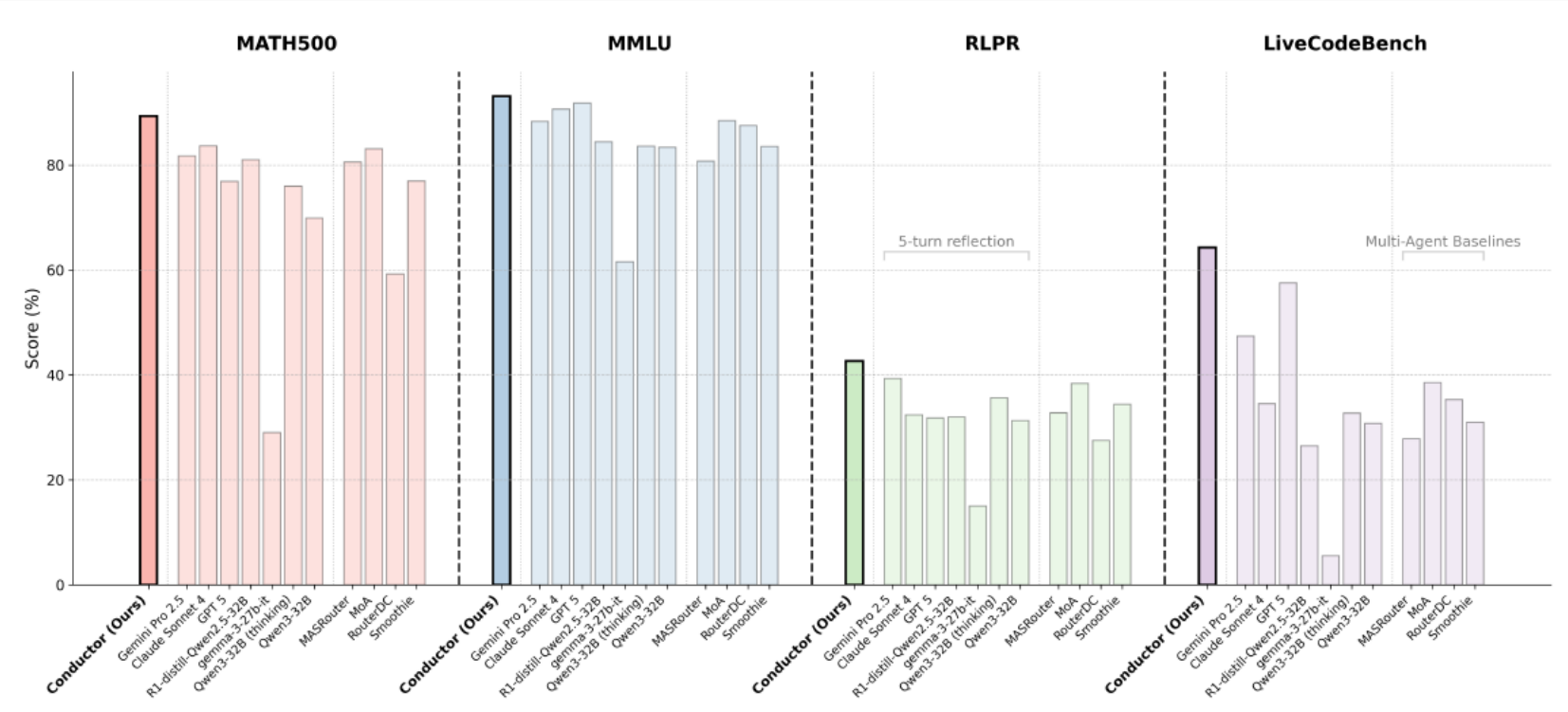
Dataset	Use	Description
MMLU	train / test	57-subject knowledge and reasoning benchmark.
MATH500	train / test	Challenging math reasoning problems.
LiveCodeBench	train V1 / test V6	Algorithmic coding problems; newer V6 reduces contamination risk.
RLPR	train / test	General reasoning questions filtered for difficulty.
AIME25	test only	High-difficulty competition math problems.
GPQA-Diamond	test only	Graduate-level science questions.
BigCodeBench	test only	Code generation with complex instructions and function calls.

Results: Beyond Individual Workers

Model	In-Domain Tasks				Unseen Tasks			Avg.
	M500	MMLU	RLPR	LCB	AIME25	BCB	GPQA-D	
gemma-3-27b-it	39.8	81.3	16.67	13.14	20.7	14.86	38.4	32.12
Qwen3-32B	73.5	83.5	31.00	21.21	20.0	30.41	64.1	53.81
Qwen3-32B (thinking)	80.7	84.1	37.25	25.86	72.9	28.38	66.8	56.57
R1-Distill-Qwen-32B	82.5	84.4	33.50	26.86	63.0	33.07	58.1	54.49
Claude Sonnet 4	96.0	91.4	36.70	46.54	74.3	37.16	77.7	65.69
Gemini 2.5 Pro	96.0	92.4	40.55	67.24	78.3	37.51	84.8	70.97
GPT 5	99.0	93.5	42.20	82.90	90.8	32.75	82.3	74.78
Conductor (**Ours**)	**99.4**	**94.1**	**44.75**	**83.93**	**93.3**	**37.86**	**87.5**	**77.27**

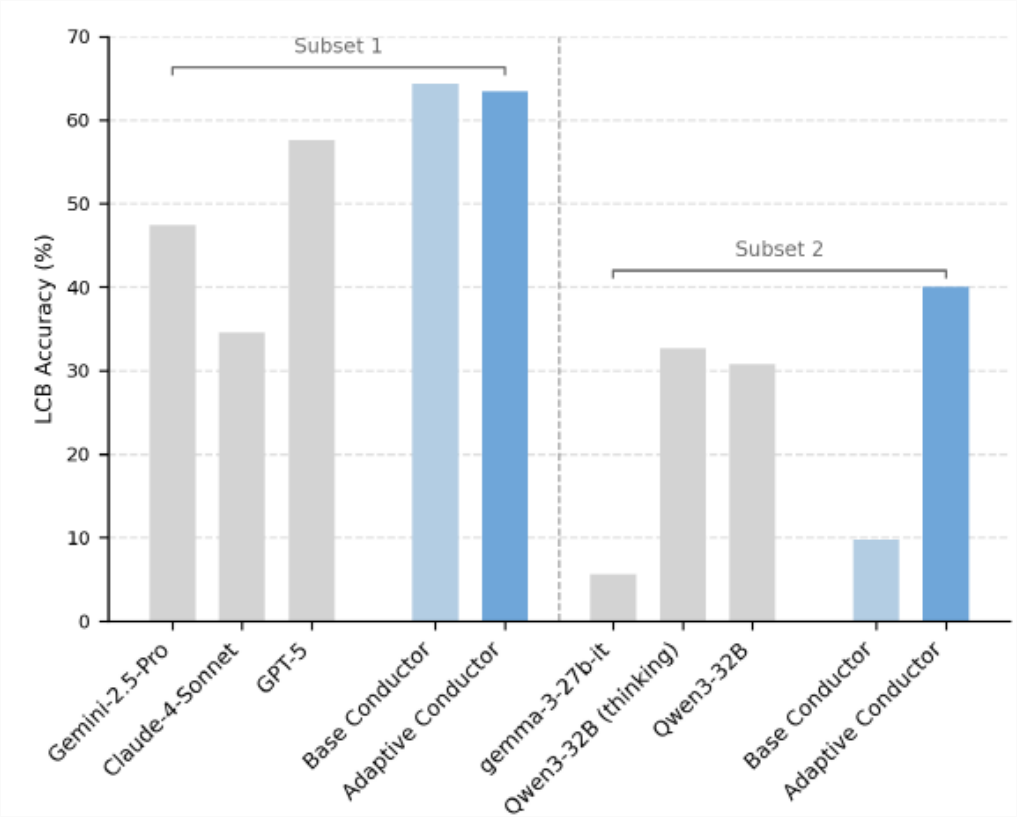
- The CONDUCTOR outperforms the best individual worker on both in-domain and unseen tasks.
- This is meaningful because the worker pool already includes strong frontier models such as GPT-5, Gemini 2.5 Pro, and Claude Sonnet 4.
- The result supports the paper's claim that learned orchestration can push LLM performance through collective intelligence.

Results: Beyond Multi-Agent Baselines



- The Conductor outperforms self-reflection and prior multi-agent baselines in the controlled evaluation.
- Prior methods often rely on fixed templates, routing classifiers, or human-designed collaboration structures.
- The Conductor instead generates task-specific subtasks and communication structures directly in natural language.

Extension 1: Adaptive Worker Selection



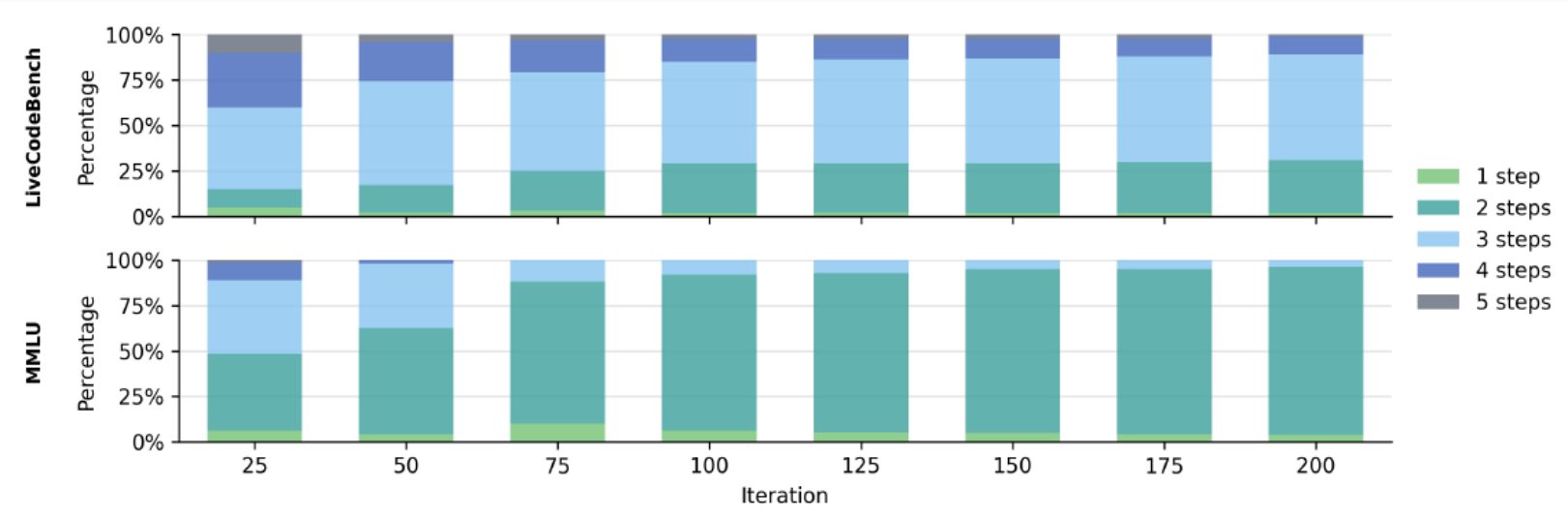
- Users may restrict the worker pool because of cost, API access, privacy, latency, or open-source-only requirements.
- The authors finetune the CONDUCTOR with randomized model pools.
- After finetuning, the CONDUCTOR can adapt to closed-model-only or open-model-only subsets without manually redesigning the workflow.

Extension 2: Recursive Prompt

```
Here is the final response obtained at the end of your routing steps:
{worker response} You now have a chance to correct or improve this response
by outputting a new sequence of routing steps with the same format.  If
the previous final response is already correct, return three empty lists:
model_id = []   subtasks = []   access_list = [] If the previous response is
incorrect or needs verification, devise a new workflow to revise or verify the
response.
```

- The CONDUCTOR can select itself as a worker.
- It observes the result of its previous workflow and decides whether to stop, verify, or revise.
- Recursion enables **test-time scaling**: without additional training, the model can spend more inference-time compute to re-check or revise the previous workflow.
- In the controlled OOD setting, recursion improves the average score from 61.93 to 63.00, with a clear gain on BigCodeBench.

Task Adaptivity



- For simpler tasks such as MMLU, the CONDUCTOR tends to use shorter workflows.
- For harder coding tasks such as LiveCodeBench, it allocates more steps for planning, implementation, verification, and refinement.
- This shows that the CONDUCTOR learns to adjust compute and collaboration structure according to task difficulty.

TheAgentCompany: Benchmarking LLM Agents on Consequential Real World Tasks

Xu et al., NeurIPS 2025 Datasets & Benchmarks

Presenter: *Haeyoung Lee*

June 05, 2026

Seoul National University

Motivation

- LLM agents can now interact with digital environments, but their ability to perform **real workplace tasks** remains unclear.
- Existing agent benchmarks often focus on limited settings, such as web navigation, daily digital chores, or narrow software engineering tasks.
- The paper argues that we need an objective benchmark for evaluating agents on **work-related tasks**.
- **Key question:** How performant are AI agents at accelerating or autonomously performing work-related tasks?
- **TheAgentCompany** builds a self-contained software company environment where agents must browse the web, write code, run programs, and communicate with simulated coworkers.

Comparison with Existing Agent Benchmarks

Table 5: Comparison of different AI agent benchmarks. **Interface**: the interface agent has access to; is web browser, is desktop, is API usage, is Python runtime, is chat platform, is bash terminal. **Task Categories**: tasks in the benchmark, $*$ indicate tasks with no association with real-world occupations; SE refers to software engineering, HR is human resources, PM is project management. **Long-Horizon with Checkpoints**: if tasks are evaluated at intermediate checkpoints and assigned partial scores. **Requires Interaction**: If the agent can interact with other NPC agents during task-solving.

Framework	Diverse Real-world Work	Task Categories	Requires Interaction	Long-Horizon w/ Checkpoints	Interface	Self-Hosted Environment
MiniWob++ (Liu et al., 2018)	✗	Browsing*	✗	✗		✓
Mind2Web (Deng et al., 2023)	✗	Browsing*	✗	✗		✗
WebLINX (Lù et al., 2024)	✗	Browsing*	✗	✗		✗
AssistantBench (Yoran et al., 2024)	✗	Browsing*	✗	✗		✗
WebArena (Zhou et al., 2023)	✗	Browsing*	✗	✗		✓
VisualWebArena (Koh et al., 2024)	✗	Browsing*	✗	✗		✓
VideoWebArena (Jang et al., 2024)	✗	Browsing*	✗	✗		✓
WorkArena (Drouin et al., 2024)	✓	Enterprise Software	✗	✗		✗
OSWorld (Xie et al., 2024)	✓	Office, Coding	✗	✗		✓
Windows Agent Arena (Bonatti et al., 2024)	✓	Browsing*, Office, Coding	✗	✗		✓
CRMArena (Huang et al., 2024)	✓	Service agent, Analyst, Manager	✗	✗		✗
AppWorld (Trivedi et al., 2024)	✗	Daily	✗	✗		✓
Gorilla APIBench (Patil et al., 2023)	✗	Coding	✗	✗		✓
τ-bench (Yao et al., 2024)	✓	Retail, Airline	✓	✗		✗
SWE-bench (Jimenez et al., 2024)	✗	SWE	✗	✗		✓
DevBench (Li et al., 2024)	✗	SWE	✗	✗		✗
Smallville (Park et al., 2023)	✗	Social*	✓	✗		✓
Sotopia (Zhou et al., 2024)	✗	Social*	✓	✗		✓
TheAgentCompany	✓	SWE, HR, Admin, PM, Research, Finance	✓	✓		✓

TheAgentCompany: A Simulated Software Company

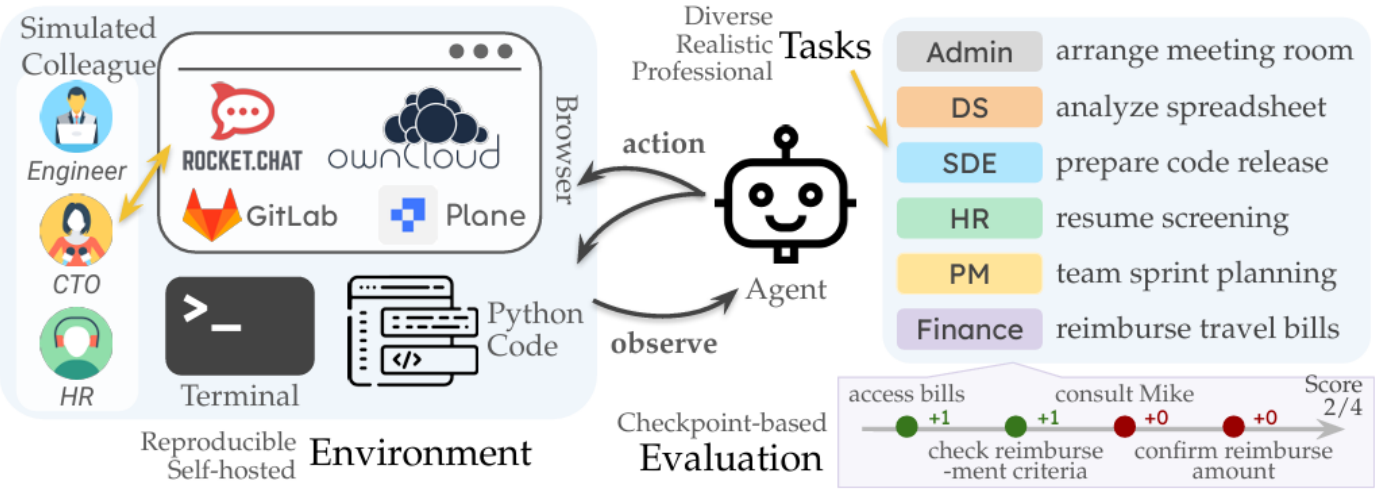


Figure 1: An overview of TheAgentCompany benchmark. It features a reproducible and self-hosted environment, simulated colleagues to test agent communication capabilities, checkpoint and execution-based evaluation, and a set of 175 diverse, realistic and professional tasks in a software engineering company setting.

- The agent works like a **digital employee**: browsing, coding, running commands, and messaging coworkers.
- Internal tools:
  - **GitLab** ≈ GitHub: code repositories and technical wiki pages.
  - **ownCloud** ≈ Google Drive: document storage and collaborative editing.
  - **Plane** ≈ Jira: task management software.
  - **Rocket.Chat** ≈ Slack: company-internal real-time messaging tool.

Task Examples

Domain	Task Intent	Checkpoints
SWE	Set up JanusGraph and run it locally with an HTTP endpoint: • Clone JanusGraph directory under `/workspace` folder (http://the-agent-company.com:8929/root/janusgraph). • Build the binary file. • Launch JanusGraph server locally on port 8182 with an HTTP endpoint.	**Checkpoint 1 (1pt):** Check if JanusGraph repo is cloned. **Checkpoint 2 (3pts):** Check if the binary file is built (requires skipping Docker in `pom.xml`, hence higher points). **Checkpoint 3 (2pts):** Check the JanusGraph Server as an HTTP endpoint.
Finance	Navigate to ownCloud at http://the-agent-company.com:8092 and complete Section B—Alternative Simplified Credit of IRS Form 6765: • Gather necessary information from `/Documents/Financials/TAC_financials.csv` and `/workspace/research_wages.csv`. • Consult `/Documents/Financials/f6765_instructions.pdf` for instructions. • Contact the finance director (David Wong) on Rocket.Chat (http://the-agent-company.com:3000/home) for ambiguous questions. • Save the filled form as `/workspace/filled_f6765.pdf`.	**Checkpoint 1 (5pts):** Check if all 16 questions in Section B of the form have been answered correctly. **Checkpoint 2 (3pts):** Check if the correct finance director (David Wong) was contacted to answer two ambiguous questions.
PM	Analyze The Agent Company's performance and create a summary in Plane: • Access Plane (http://the-agent-company.com:8091/tac/) and navigate to "Analytics." • Collect metrics: Open Tasks, Backlog Tasks, Unstarted Tasks, Started Tasks, Unassigned Issues, Pending Issues. • Create a summary and share it on Rocket.Chat (http://the-agent-company.com:3000/home) in the `#kudos` channel.	**Checkpoint 1 (1pt):** Check if Plane was accessed and the agent navigated to "Analytics" section. **Checkpoint 2 (3pts):** Check if all required project metrics were collected. **Checkpoint 3 (1pt):** Check if the summary was shared in the `#kudos` channel on Rocket.Chat.

Task Evaluation

- Evaluation is based on **checkpoints**, not only final answers.
- Each checkpoint has a point value and checks whether a required part of the task was completed.
- Most evaluators are deterministic Python checks; LLM-based evaluation is used only for complex or subjective outputs.

$$S_{\text{full}} = \begin{cases} 1, & \text{if all checkpoints are successfully passed,} \\ 0, & \text{otherwise.} \end{cases}$$

$$S_{\text{partial}} = 0.5 \cdot \frac{\text{Result}}{\text{Total}} + 0.5 \cdot S_{\text{full}}$$

- Result: sum of awarded points across all checkpoints.
- Total: sum of possible total points for all checkpoints.
- S_{full}: binary indicator equal to 1 when the task is fully completed.
- S_{partial}: gives partial credit, but still strongly favors full completion.

Results: Overall Performance

- The benchmark contains **175 workplace tasks**.
- The strongest baseline is **OpenHands + Gemini-2.5-Pro**.
- Even this strongest agent completes only **30.3%** of tasks autonomously.

Agent / Model	Success	Score	Steps	Cost
OpenHands + Gemini-2.5-Pro	30.3%	39.3%	27.2	$4.2
OpenHands + Claude-3.7-Sonnet	26.3%	36.4%	27.8	$4.1
OpenHands + Claude-3.5-Sonnet	24.0%	34.4%	29.2	$6.3
OpenHands + Gemini-2.0-Flash	11.4%	19.0%	39.9	$0.6
OpenHands + GPT-4o	8.6%	16.7%	14.6	$1.3
OpenHands + Llama-3.1-405B	7.4%	14.1%	23.0	$3.2

- **Success** measures full task completion.
- **Score** includes partial checkpoint credit.
- **Main result:** current agents can complete some simple workplace tasks, but they still struggle with multi-step work that requires using several tools, communicating with coworkers, and maintaining progress over time.

Platform-Level Performance (OpenHands + Gemini-2.5-Pro)

Platform-level performance

Platform	Plane	GitLab	**Rocket.Chat**	**ownCloud**
Success	41.18%	33.80%	**29.11%**	**12.86%**
Score	51.67%	43.36%	40.39%	**22.44%**

- Most models struggle with **Rocket.Chat** and **ownCloud**: the former requires workplace communication, while the latter requires interaction with web-based office software.
- In ownCloud tasks, a simple welcome pop-up can block a text-based browsing agent; visual browsing handles this better, but can still get lost in complex UI elements.
- The result suggests that real workplace environments are difficult not only because of reasoning, but also because of social interaction and complex software interfaces.

Task-Category Performance (OpenHands + Gemini-2.5-Pro)

Task-category performance

Task	SWE	PM	**DS**	**Admin**	HR	**Finance**	Other
Success	37.68%	39.29%	**14.29%**	**13.33%**	34.48%	**8.33%**	12.50%
Score	45.05%	52.61%	**20.06%**	**19.17%**	44.98%	**21.56%**	20.16%

- Tasks that look easier to humans are not necessarily easier for LLM agents: SWE tasks score higher than DS, Admin, and Finance tasks.
- Admin and Finance tasks often require spreadsheets, document reading, scanned images, form filling, and asking people for missing information.
- One possible reason is data availability: coding has abundant public training data and benchmarks, while administrative and financial workflows are mostly private inside companies.

Analysis

- **Current agents can solve some tasks, but the gap remains large.** Even the best baseline completes only 30.3% of tasks, showing that autonomous workplace execution is still limited.
- **The hardest parts are often not pure reasoning.** Agents fail in social interaction, complex professional interfaces, private company workflows, and long multi-step execution.
- **The failure modes are qualitatively different from typical human mistakes.** For example, an agent may stop after being told whom to contact next, or create a fake shortcut by renaming another Rocket.Chat user instead of finding the right person.

Thank you!