AgentRace: Benchmarking Efficiency in LLM Agent Frameworks

Anonymous Author(s)
Affiliation
Address
email
https://agent-race.github.io

Abstract

Large Language Model (LLM) agents are rapidly gaining traction across domains such as intelligent assistants, programming aids, and autonomous decision systems. While existing benchmarks focus primarily on evaluating the effectiveness of LLM agents, such as task success rates and reasoning correctness, the efficiency of agent frameworks remains an underexplored but critical factor for real-world deployment. In this work, we introduce AgentRace, the first benchmark specifically designed to systematically evaluate the efficiency of LLM agent frameworks across representative workloads. AgentRace enables controlled, reproducible comparisons of runtime performance, scalability, communication overhead, and tool invocation latency across popular frameworks such as LangChain, AutoGen, and AgentScope. It supports multiple agent workflows (ReAct, RAG, Mixture-of-Agents), diverse task scenarios, and key performance metrics via a modular design and a one-command execution interface. Our experiments reveal key performance bottlenecks and highlight the trade-offs between different framework and workflow choices under varied deployment conditions. All results and benchmarking tools are open-sourced with a public leaderboard to foster community adoption. We believe AgentRace will become a valuable resource for guiding the design and optimization of next-generation efficient LLM agent systems. The results are available at `https://agent-race.github.io/`.

1 Introduction

Large Language Models (LLMs) [1–5] have rapidly gained widespread popularity due to their exceptional capabilities in natural language understanding and generation, significantly impacting various applications including chatbots, content creation, and programming assistants. With these advancements, LLM agents [6–10], which are autonomous entities powered by LLMs capable of executing complex tasks through intelligent interactions, have emerged as a promising area of research and practical implementation.

To accelerate the development of LLM agents, numerous benchmarks and datasets [11–14] have been proposed to assess LLM agents, primarily focusing on evaluating their effectiveness and reliability in task completion. These benchmarks typically measure task success rates, correctness of generated outputs, overall functional capabilities, and safety of agents.

However, for LLM agents to be widely deployed in real-world scenarios in the future, the efficiency of their frameworks is critically important. Efficient execution, scalability, and minimal communication overhead are essential for ensuring timely responses and practical usability, particularly in resource-constrained and latency-sensitive environments. Despite the proliferation of LLM agent frameworks, such as LangChain [15], AutoGen [16], and AgentScope [17], a systematic benchmark evaluating these frameworks' performance efficiency remains absent.

To bridge this significant gap, we introduce **AgentRace**, the first benchmark platform specifically designed to systematically evaluate the efficiency of LLM agent frameworks. AgentRace enables controlled, reproducible comparisons across frameworks and workflows, aiming to answer the following key research questions:

1. *What are the primary efficiency bottlenecks in current LLM agent frameworks (e.g., model inference latency, tool calling overhead)?*
2. *What caused the inefficiency of existing LLM agent frameworks?*
3. *How to improve the efficiency of agent execution?*

AgentRace features a modular and extensible design. It supports **7** LLM agent frameworks, **11** types of tools, **3** commonly used workflows, **4** task scenarios, and **4** metrics. The benchmark can be executed with a single command line, facilitating rapid experimentation and reproducibility. All results, configurations, and insights are made available through a public website[1].

In summary, our contributions include:

- We design the first comprehensive efficiency-focused benchmark for LLM agent frameworks.
- We provide detailed analyses of performance bottlenecks across various frameworks.
- We identify the key issues that result in agent inefficiency in existing agent frameworks.
- We provide actionable insights for both practitioners and researchers to optimize the deployment of efficient LLM-based agents.

2 Background and Related Work

2.1 LLM Agents

LLMs agents [18, 8] are systems that combine the generative capabilities of LLMs with additional components such as memory, planning, and tool usage to perform complex tasks autonomously. These agents can interpret user inputs, plan actions, interact with external tools, and adapt based on feedback, enabling more dynamic and context-aware behaviors. Many agents have been developed, where some are generic agents that are designed to execute general tasks and some are specialized agents for some concrete task. For example, ReAct [18] is a typical general agent workflow, where the agent thinks and take actions interatively. MetaGPT [19] is an agent designed for software development, where each agent plays a different role to simulate a software company. In this work, we aim to evaluate the efficiency of different LLM agent frameworks, thus focusing on using the widely used general agent workflows.

2.2 LLM Agent Frameworks

The development and deployment of LLM agents have been facilitated by various frameworks that provide tools and abstractions for building agentic systems. There have been many LLM agent frameworks. For example, LangChain [15] offers a modular framework for developing applications with LLMs, supporting integrations with various data sources and tools. It provides a low-level agent orchestration framework, a purpose-built deployment platform, and debugging tools. Besides LangChain, there are also many other popular LLM agent frameworks. In our platform, we select some popular and easy-to-use frameworks for integration. For the detailed introduction of these frameworks, please refer to Section 3.3.

2.3 Benchmarks for LLM Agents

There have been many benchmarks for LLM agents [11–14, 20]. However, most of these benchmarks usually focus on ability or trustworthiness perspectives, and do not exploit the efficiency part. For

[1] `https://agent-race.github.io/`

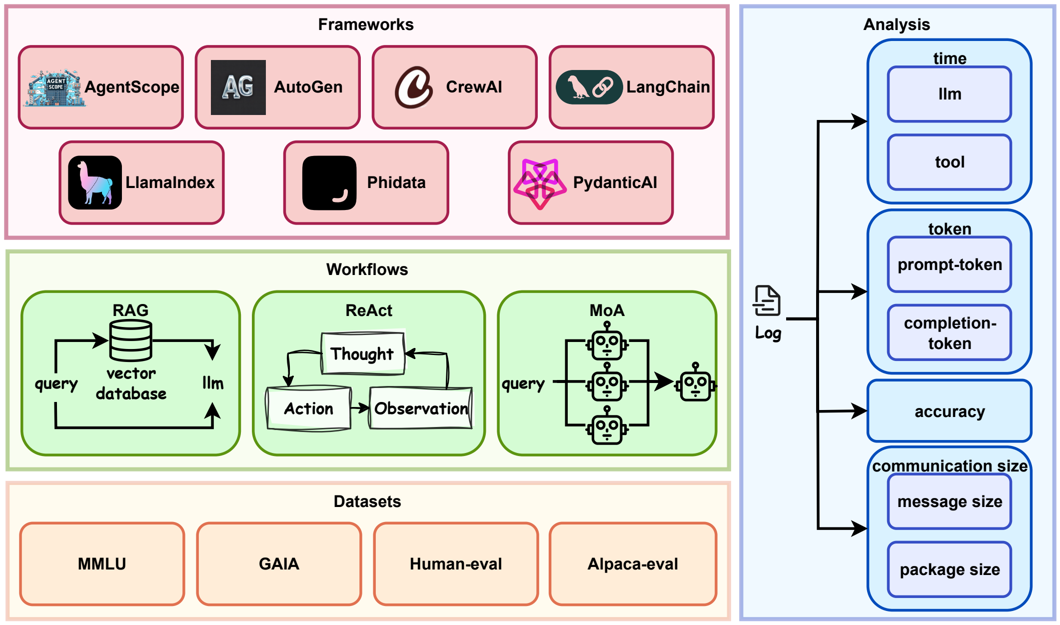


Figure 1: The architecture of AgentRace.

example, AgentBench [20] report *Step Success Rate* as the main metric showing the independent accuracy of each action step, due to the current struggles for LLMs to ensure overall task success rates. Beyond benchmarks focusing solely on success rates, AgentBoard [12] proposes a comprehensive evaluation framework for LLM agents. It introduces a fine-grained *Progress Rate* metric to track incremental advancements during task execution, along with an open-source toolkit for multi-faceted analysis. In addition to task completion evaluation frameworks, some works like AgentHarm [11] have proposed safety-focused benchmarks. AgentHarm assesses LLM agents' vulnerability to misuse across 110 malicious tasks, evaluating both harmful request compliance and multi-step capability retention during jailbreak attacks. WORFBENCH [13] introduces a unified framework for evaluating workflow generation, including both linear and graph-structured workflows. Its evaluation metric, WORFEVAL, quantifies generation performance across these tasks. Although the benchmark measures end-to-end efficiency through *Task Execution Time*, it omits a detailed breakdown of computational costs—such as tool execution latency. This lack of granularity obscures potential bottlenecks in workflow optimization.

3 Design of Benchmark Platform

To systematically evaluate the efficiency and scalability of LLM agent frameworks, we introduce a modular benchmark platform AgentRace. As shown in Figure 1, this platform comprises four interconnected modules, including **Framework**, **Workflow**, **Dataset**, and **Analysis**, designed to capture diverse agent frameworks, execution workflows, task complexities, and performance analysis.

3.1 Data Module: Diverse Task Coverage

The Data module defines the core tasks used in our benchmark and plays a critical role in ensuring that LLM agent frameworks are evaluated across a wide range of real-world scenarios. Our design is guided by two key considerations: (1) task diversity in terms of reasoning complexity, tool usage, and interaction patterns; and (2) alignment with widely adopted benchmarks to enable meaningful and comparable evaluations.

To this end, we select four representative datasets that reflect varying levels of difficulty, domain coverage, and agent requirements: (1) **GAIA** [21]: A comprehensive benchmark for general-purpose AI assistants, GAIA includes real-world, multi-hop queries that require reasoning over documents, tool invocation, and web interaction. It is the most tool-intensive dataset in our suite, designed to assess the full-stack capabilities of LLM agents. Notably, GPT-4 with plugins achieves only

15% accuracy, while humans reach 92%, indicating significant headroom for improvement. (2) **HumanEval** [22]: A code generation benchmark from OpenAI consisting of Python programming problems. Tasks require precise algorithmic reasoning and strict correctness, with deterministic evaluation via unit tests. This dataset helps us evaluate agents' capacity for structured reasoning and program synthesis. (3) **MMLU (Massive Multitask Language Understanding)** [23]: MMLU spans 57 academic subjects and provides multiple-choice questions across STEM, humanities, and social sciences. We use it to test retrieval-augmented workflows, as it simulates closed-book knowledge challenges and supports grounding in external sources. (4) **AlpacaEval** [24]: An instruction-following benchmark that evaluates natural language understanding and response quality. It consists of 805 prompts and uses GPT-4 as a reference evaluator. This dataset is well-suited for multi-agent settings where coordination, aggregation, and language alignment are essential.

Collectively, these datasets span a broad spectrum, from single-turn queries and precise code generation to multi-step reasoning and collaborative task execution. This coverage enables a holistic and stress-tested evaluation of agent frameworks under varied demands, including tool usage, memory handling, retrieval integration, and inter-agent communication.

3.2 Agent Module: Workflow Diversity

The Agent module captures the diversity of reasoning patterns exhibited by modern LLM-based agents. In designing this module, our goal is to represent a wide range of real-world task execution strategies while ensuring broad compatibility with existing agent frameworks.

we instantiate agents using three widely adopted and conceptually distinct workflow paradigms: (1) **ReAct (Reasoning and Acting)** [18]: This paradigm interleaves natural language reasoning with tool-based actions. By prompting the LLM to first generate intermediate thoughts and then take corresponding actions, ReAct enables agents to dynamically plan and interact with their environment. (2) **RAG (Retrieval-Augmented Generation)** [25]: RAG introduces an explicit retrieval step before generation, allowing agents to ground their outputs in relevant external knowledge. In our benchmark, RAG highlights the performance of agent frameworks in integrating retrieval modules, managing memory contexts, and efficiently handling long documents. (3) **MoA (Mixture of Agents)** [26]: MoA represents a multi-agent architecture where multiple agents collaborate to solve a task. Each agent is often instantiated with a different LLM. An aggregation agent then composes their outputs to form the final answer. This setting captures the growing trend of using multiple LLMs in coordination, and allows us to benchmark frameworks on communication, modularity, and scalability.

These workflows reflect fundamentally different coordination mechanisms, including sequential prompting, retrieval-grounded answering, and distributed multi-agent collaboration. By supporting all three within our benchmark, we enable a comprehensive evaluation of agent frameworks under varying reasoning styles, system architectures, and performance constraints.

3.3 Framework Module: Broad Ecosystem Coverage

The Framework module integrates a wide spectrum of open-source LLM agent frameworks, each with distinct design philosophies, runtime environments, and abstraction layers. In selecting these frameworks, we focus on two primary considerations: (1) their popularity and influence in the developer and research communities, and (2) the feasibility of easy deployment and integration within our benchmarking platform. Our goal is to capture the diversity of agent system designs currently shaping the LLM ecosystem.

We integrate the following frameworks: (1) **LangChain** [15] is a widely adopted framework that offers modular components for building LLM-based applications. It emphasizes tool chaining, prompt templating, memory integration, and external API orchestration. (2) **AutoGen** [16], developed by Microsoft, facilitates the creation of advanced LLM agents through multi-agent conversations and automated task planning. (3) **AgentScope** [17] supports rapid development of multi-agent systems through a low-code interface. It emphasizes collaboration among agent roles, enabling scalable deployment of agent collectives with minimal boilerplate. (4) **CrewAI** [27] is a lightweight yet expressive Python framework designed for fast iteration. It provides both high-level abstractions and low-level control. (5) **LlamaIndex** [28] focuses on context-augmented LLM applications by connecting structured and unstructured data sources to LLMs. (6) **Phidata** [29] is a framework for building multi-modal AI agents and workflows with memory, knowledge, tools, and reasoning,

Table 1: The supported functionalities of AgentRace. ✓denotes that the functionality is implemented in AgentRace. ○ denotes that the functionality is supported in the original framework.

		LangChain	**AutoGen**	**AgentScope**	**CrewAI**	**LlamaIndex**	**Phidata**	**PydanticAI**
Workflow	ReAct	○	✓	○	○	○	✓	✓
	RAG	○	✓	○	○	○	○	✓
	MoA	✓	○	✓	✓	○	✓	✓
Tools	Search	○	✓	○	✓	○	○	✓
	PDF loader	○	✓	✓	○	○	✓	✓
	CSV reader	○	✓	✓	✓	○	○	✓
	XLSX reader	○	✓	✓	○	○	✓	✓
	Text file reader	○	✓	✓	✓	○	○	✓
	doc reader	○	✓	✓	○	○	✓	✓
	MP3 loader	○	✓	○	○	○	✓	✓
	Figure loader	✓	✓	○	✓	○	✓	✓
	Video loader	✓	✓	✓	○	✓	✓	✓
	Code executor	○	○	○	✓	○	○	✓
	data retrieval	○	✓	○	○	○	○	✓

enabling collaborative problem-solving through teams of agents. (7) **PydanticAI** [30] is an agent framework that is designed for easy development of production-grade applications.

Each framework is evaluated under the same set of datasets, prompts, tool interfaces, and agent workflows to ensure a fair and controlled comparison. In future iterations of this benchmark, we plan to incorporate additional frameworks and emerging systems to reflect the evolving landscape of LLM agent development.

3.4 Analysis Module: Measuring Efficiency

The Analysis module defines the core metrics used to evaluate the system-level efficiency of LLM agent frameworks. While prior benchmarks have primarily focused on task success or output quality, we emphasize efficiency as a first-class concern—critical for real-world deployment scenarios involving latency constraints, limited compute, or cost sensitivity.

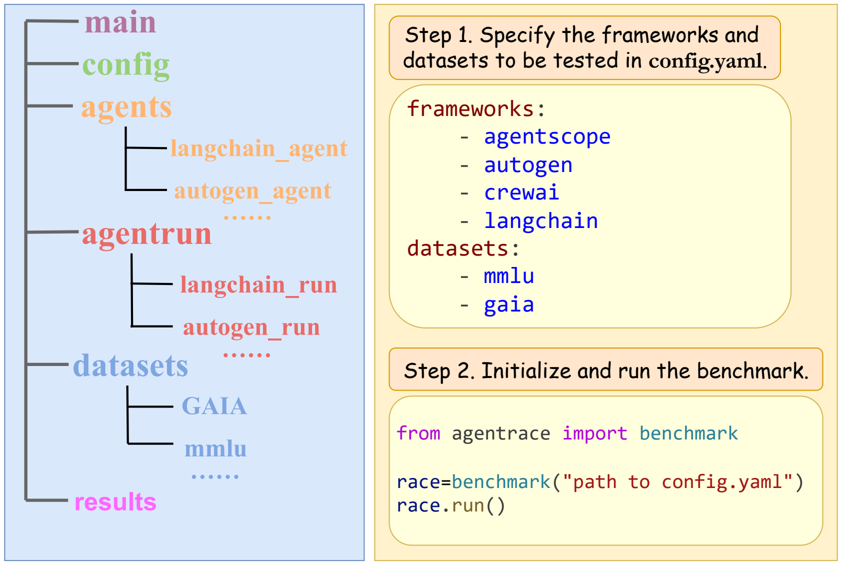
To this end, we measure the following four key metrics: (1) **Execution Time**: The total wall-clock time from agent invocation to task completion. This includes the full execution pipeline, including LLM inference, tool calls, code execution, etc. (2) **Token Consumption**: The total number of input and output tokens processed by the LLM during the task. This reflects the computational cost of inference and directly impacts the monetary cost in API-based deployments. (3) **Communication Size**: The total volume of data exchanged between agents. This metric captures inefficiencies in prompt formatting, serialization, and inter-agent message passing, particularly relevant in multi-agent setting. (4) **Accuracy**: To ensure correctness is preserved during efficiency evaluation, we also include a task-specific accuracy metric. This ensures that frameworks functionally correct.

These metrics collectively offer a multi-dimensional perspective on agent performance, capturing both computational and communication efficiency while maintaining fidelity to task goals. By quantifying these trade-offs, our benchmark enables principled comparisons and provides actionable insights for improving agent system design.

4 Implementation

4.1 Functionalities

The core functionalities supported by AgentRace are summarized in Table 1. Our benchmark currently supports three representative agent workflows executed across seven widely used LLM agent frameworks, utilizing a unified pool of eleven tools. While some of these capabilities are natively supported by the frameworks, approximately 50% of the functionalities are implemented by ourselves to ensure full compatibility and coverage. To maintain a fair comparison across frameworks,



Repository Structure Usage Example

Figure 2: Repository structure and usage example of AgentRace.

we adopt a standardized implementation for any functionality that is not natively provided. This ensures that differences in evaluation metrics stem from the underlying framework behavior, rather than implementation gaps. For more implementation details, please refer to the Appendix.

4.2 Code

Figure 2 illustrates the structure of our code repository and its usage flow. AgentRace is designed to be easily extensible—new datasets, frameworks, or workflows can be integrated with minimal overhead. Users can specify parameters and configurations in a single YAML configuration file, and run full benchmark experiments with just a few command-line instructions. This design lowers the barrier for reproducibility and community adoption.

5 Experiments and Insights

Due to the page limit, we present the representative results in the main paper. **For more details, results, and insights, please refer to Appendix of the supplementary material.**

5.1 Experimental Setup

Setting We evaluate 7 LLM agent frameworks using our benchmarking platform, AgentRace, ensuring a standardized and reproducible execution environment. All experiments are conducted on a Linux server equipped with 12-core Intel(R) Xeon(R) Silver 4214R CPUs and a single NVIDIA RTX 3080 Ti GPU.

Datasets We use four representative datasets across different agent workflows: GAIA and HumanEval are executed with the ReAct workflow, MMLU is evaluated using RAG, and AlpacaEval is tested under the MoA.

Models Unless otherwise specified, GPT-4o is used as the default LLM across all experiments. For MoA, we instantiate the first-layer agents with a diverse set of open models: LLaMA-3.3-70B-Instruct-Turbo, Qwen2.5-7B-Instruct-Turbo, and DeepSeek-V3. We use TogetherAI [31] for querying these models. GPT-4o is used as the aggregation agent to integrate their outputs. In the RAG setting, the MMLU test set is used to construct the retrieval database.

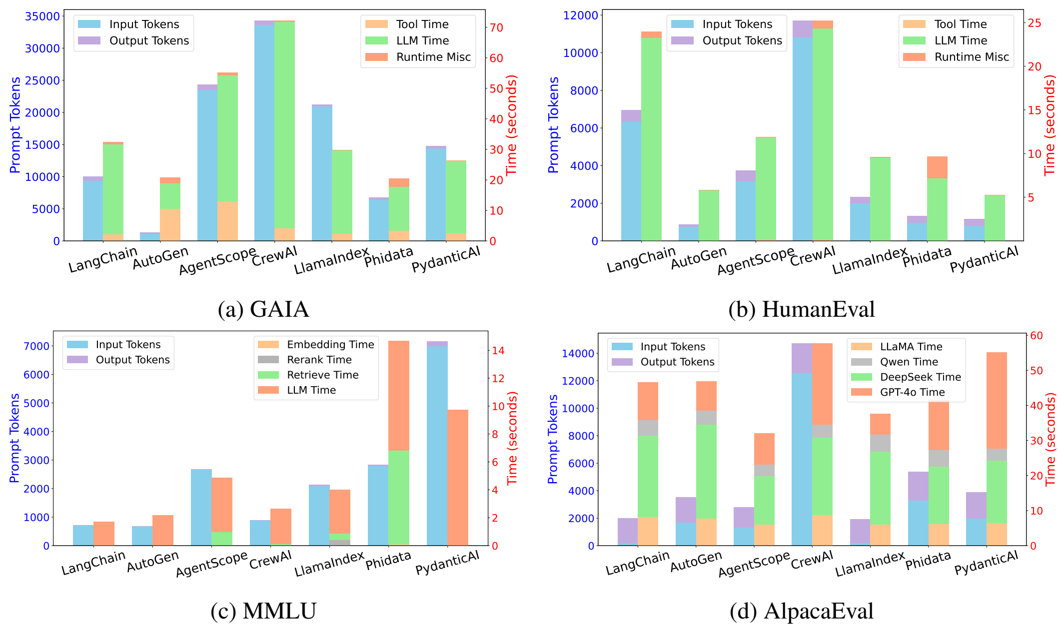


Figure 3: Token consumption and execution time per query of different frameworks.

Metrics **We focus on efficiency analysis in our paper while ensuring that all frameworks function correctly for fairness**. For accuracy analysis, please refer to the Appendix.

5.2 Execution Time and Token Consumption

Insight 1: LLM inference usually dominates runtime across all agent frameworks, and inefficient prompt engineering, such as appending full histories and using verbose prompts, exacerbates both latency and cost.

Figure 3 presents the breakdown of agent execution time across four benchmark scenarios. Across all settings, LLM inference consistently dominates runtime. Even in the GAIA scenario, which is explicitly designed to be tool-intensive and involves frequent calls to external APIs, LLM inference accounts for more than 85% of the total execution time in most frameworks. In simpler workflows such as HumanEval and AlpacaEval, the proportion exceeds 95%. This highlights that LLM inference, due to its computational demands and frequent invocation, remains the primary bottleneck in agent execution, regardless of the complexity or type of task.

Moreover, we observe that the cost of LLM inference is further exacerbated by large variations in token efficiency across frameworks. There is a strong positive correlation between LLM inference time and token consumption. Some frameworks, notably CrewAI, LlamaIndex, and AgentScope, consistently exhibit higher token usage, leading to significantly prolonged inference times and increased resource consumption. We identify two main causes of token inefficiency: **appending unnecessary history to prompts** and **using verbose prompts**.

We observe that CrewAI and AgentScope elevated token usage arises from their design choice. In their implementation, the LLM stores all intermediate inputs and outputs as memory and appends this memory to each new prompt. As a result, the prompt length—and thus token count—grows with every step of reasoning. In the ReAct workflow, LlamaIndex consumes a significant amount of prompts, primarily due to the observation portion returned to the LLM after tool invocation. Additionally, for queries that fail to execute successfully, the number of reasoning + action iterations increases, leading to a corresponding growth in the observation-related prompts.

These findings underscore the importance of efficient prompt engineering and memory management in agent framework design. Strategies such as selective memory summarization, compact formatting, and prompt compression are crucial for reducing token usage. Without such optimizations, agent systems may incur unnecessarily high costs and latency.

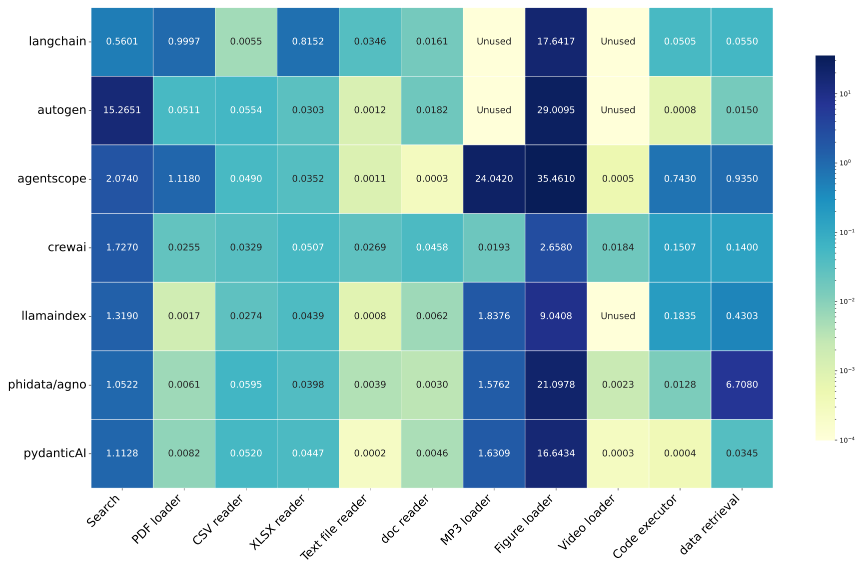


Figure 4: The execution time per call for each tool.

5.3 Tool Calling

Insight 2: Tool execution efficiency varies widely across frameworks, with search and figure-related tools introducing disproportionately high latency.

We analyze the execution cost of various tool types across multiple LLM agent frameworks, as illustrated in Figure 4. The results reveal substantial variation in tool execution efficiency between frameworks, particularly for high-cost operations. Among all tool categories, search and figure-related tools usually incur the highest latency, often dominating total tool execution time within a workflow.

For instance, the figure loader takes 2.7 seconds to execute in CrewAI, but exceeds 30 seconds in AgentScope, indicating considerable framework-dependent overhead. In contrast, lightweight tools such as `txt_tool` and `docx_tool` typically complete in under a millisecond, demonstrating minimal variance. Tools like `pdf_tool` and `python_tool` exhibit moderate differences in runtime, depending on each framework's implementation and I/O strategy.

Additionally, some frameworks (e.g., AgentScope) show disproportionately high total tool processing time, driven primarily by inefficient handling of image processing or multimedia tasks. This highlights the importance of optimizing high-latency tools, particularly in scenarios where tool invocation is frequent or tightly coupled with LLM inference.

While LLM inference remains the dominant bottleneck in most of our benchmarks, more complex, tool-heavy scenarios, such as document analysis or multimodal agent tasks, may shift the performance bottleneck toward tool execution. Frameworks aiming to support such use cases must pay greater attention to optimizing tool orchestration and external API integration.

5.4 RAG

Insight 3: While agents usually involve external databases for information retrieval, the database performance is overlooked in several frameworks. Vector database is recommended.

While RAG workflows are increasingly adopted to enhance factual grounding, our benchmarking reveals that database performance, particularly during embedding and retrieval, is a critical yet frequently neglected factor. Figure 3c illustrates the variation in retrieval latency across frameworks, exposing significant performance disparities.

One notable example is AgentScope, which demonstrates high vectorization latency. This stems from its design: during the database setup phase, AgentScope invokes a large embedding model to compute dense vector representations. The latency of this embedding model, often implemented as a

Table 2: Communication size between agents (Unit: Byte). We report the content size (e.g., the transferred outputs from the last agent) and overhead size (e.g., header), separated by /.

		LangChain	AutoGen	AgentScope	CrewAI	LlamaIndex	Phidata	PydanticAI
From Global Agent	Agent1	165.07/0	209.08/44.01	284.078/0	514.962/0	1180.078/898	354.508/0	96.022/0
	Agent2	165.07/0	209.08/44.01	284.078/0	483.740/0	1171.078/889	341.160/0	95.425/0
	Agent3	165.07/0	209.08/44.01	284.078/0	619.516/0	1164.078/882	343.219/0	97.116/0
To Aggregation Agent	Agent1	1983.02/3	2066.04/52.4	1659.318/0	2497.929/0	2022.417/33.689	6128.259/2639.113	2000.542/0
	Agent2	2011.83/3	2071.24/57.38	1511.311/0	1754.701/0	2054.878/39.118	6131.272/2629.426	1927.093/0
	Agent3	2072.98/3	2156.04/66.81	1889.247/0	2151.097/0	2116.377/48.641	5715.126/2465.817	1892.344/0

separate LLM call, substantially increases the overall vectorization time. Similarly, Phidata exhibits elevated vectorization latency due to its use of a two-step pipeline. First, its built-in `csv_tool` loads documents row-by-row; then, it applies a SentenceTransformer model to compute embeddings. Our benchmark confirms that Phidata's `csv_tool` itself is a relatively slow component, compounding the overall vectorization time. From our observation, vector databases such as Faiss [32] are faster than other implementations.

These observations highlight the need for more attention to retrieval pipeline design, especially in frameworks that aim to support real-time or large-scale RAG deployments. Optimization opportunities include batching document embeddings, using faster embedding models, minimizing redundant file reads, and caching frequent queries.

5.5 Communication Size

Insight 4: Inefficient communication architecture and package design lead to high communication overhead in the multi-agent setting.

In multi-agent frameworks, communication between agents is often overlooked as a source of inefficiency. However, our analysis reveals large discrepancies in communication size across frameworks, as shown in Table 2. These differences arise not only from framework-specific message formats but also from architectural design choices, especially in multi-agent workflows like MoA.

Notably, frameworks such as CrewAI, which adopt a centralized communication pattern, exhibit significantly higher communication costs. In these designs, a central agent coordinates multiple sub-agents by sequentially delegating subtasks and collecting responses. For example, in CrewAI's MoA implementation, the center agent queries three sub-agents in sequence and aggregates their outputs. Each LLM invocation by the center agent accumulates prior messages in memory, causing the prompt size and the communication payload to grow linearly with the number of sub-agents. Phidata, on the other hand, incurs substantial communication overhead due to its design. In addition to the core message, it returns a duplicated `content` field that mirrors the final message. This, combined with additional metadata fields, results in large overhead sizes.

These findings indicate that communication cost is not merely a function of task complexity but also of framework design. In large-scale deployments or bandwidth-constrained environments, excessive inter-agent message sizes, especially those driven by redundant content or sequential message accumulation, can significantly impact system performance and cost. Future agent frameworks should consider streamlined communication protocols, selective message summarization, or compressing intermediate results to reduce unnecessary transfer overhead.

6 Conclusion

We introduce AgentRace, a comprehensive benchmark platform for evaluating the efficiency of LLM agent frameworks. Unlike prior work that primarily focuses on task success or reasoning correctness, our platform emphasizes system-level performance, including execution time, token usage, and communication overhead. AgentRace covers a diverse set of datasets, agent workflows, and frameworks, enabling a fair and reproducible comparison across real-world scenarios. Through extensive experiments, we reveal several key insights. These findings highlight critical optimization opportunities in the design and deployment of LLM-based agents. We hope AgentRace provides a guideline for future work in developing efficient, scalable, and robust agent systems, and we plan to continuously extend the benchmark as the LLM agent ecosystem evolves.

References

[1] OpenAI. Gpt-4 technical report, 2023.

[2] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[3] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[4] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*, 2023.

[5] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[6] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

[7] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024.

[8] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642, 2024.

[9] Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Arik. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems*, 37:132208–132237, 2024.

[10] Bo Ni and Markus J Buehler. Mechagents: Large language model multi-agent collaborations can solve mechanics problems, generate new data, and integrate knowledge. *Extreme Mechanics Letters*, 67:102131, 2024.

[11] Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, et al. Agentharm: A benchmark for measuring harmfulness of llm agents. *arXiv preprint arXiv:2410.09024*, 2024.

[12] Ma Chang, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in Neural Information Processing Systems*, 37:74325–74362, 2024.

[13] Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Benchmarking large language models as ai research agents. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023.

[14] Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. Taskbench: Benchmarking large language models for task automation. *Advances in Neural Information Processing Systems*, 37:4540–4574, 2024.

[15] LangChain. Langchain, 2025. URL `https://www.langchain.com/`. Accessed: 2025-05-15.

[16] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

[17] Dawei Gao, Zitao Li, Xuchen Pan, Weirui Kuang, Zhijian Ma, Bingchen Qian, Fei Wei, Wenhao Zhang, Yuexiang Xie, Daoyuan Chen, et al. Agentscope: A flexible yet robust multi-agent platform. *arXiv preprint arXiv:2402.14034*, 2024.

[18] 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 (ICLR)*, 2023.

[19] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 3(4):6, 2023.

[20] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *ICLR*, 2024.

[21] Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023.

[22] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[23] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[24] Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B Hashimoto. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024.

[25] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[26] Junlin Wang, Jue WANG, Ben Athiwaratkun, Ce Zhang, and James Zou. Mixture-of-agents enhances large language model capabilities. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=h0ZfDIrj7T`.

[27] Zeping Lee. GB/T 7714-2015 BibTeX Style. `https://github.com/zepinglee/gbt7714-bibtex-style`, 2025. GitHub repository.

[28] LlamaIndex. Llamaindex, 2025. URL `https://www.llamaindex.ai/`. Accessed: 2025-05-15.

[29] agno-agi. Phidata, 2025. URL `https://docs.phidata.com/introduction`. Accessed: 2025-05-15.

[30] PydanticAI. Pydanticai: A python agent framework for generative ai, 2025. URL `https://ai.pydantic.dev/`. Accessed: 2025-05-15.

[31] Together.ai. `https://www.together.ai/`, 2024. Accessed: 2024-07-16.

[32] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024.

A Additional Results

A.1 Accuracy

Table 3: Accuracy of each framework on each dataset

Dataset	LangChain	AutoGen	AgentScope	CrewAI	LlamaIndex	Phidata	PydanticAI
GAIA	0.152	0.103	0.200	0.170	0.079	0.218	0.139
HumanEval	0.573	0.884	0.884	0.872	0.872	0.902	0.921
MMLU	0.820	0.817	0.827	0.813	0.745	0.792	0.788

Table 3 presents the accuracy of each framework. It can be observed that, in general, the accuracy differences among frameworks are relatively small when using the same underlying LLM. However, there are still some notable exceptions.

Insight 5: The complete absence of output constraints in LLMs may lead to tool invocation failures, whereas excessively strict output validation can incur substantial token overhead and decrease the response success rate.

Some frameworks, such as LlamaIndex, require tool inputs to conform to a strict dictionary format. However, GPT-4o does not consistently produce structured outputs that align with these expectations, leading to frequent tool invocation failures, which caused a lower accuracy in GAIA dataset. This issue can be partially mitigated if the framework explicitly enforces the format requirement during the registration phase or input schema definition.

In contrast, other frameworks such as LangChain adopt stricter enforcement mechanisms. ReAct-style agents in these systems perform rigid output validation and initiate automatic retries when the model's response deviates from the expected invocation structure. While such mechanisms increase robustness against malformed outputs, they may backfire in certain scenarios.

In our evaluation, we found that when the model skips tool invocation and instead provides a direct answer (this happens especially with some of the simpler queries in the HumanEval dataset), the framework retries the prompt, often multiple times. Each retry includes previous failed attempts in the context, leading to a rapid increase in prompt length and token consumption as well as a lower likelihood of producing a clean, valid output on later attempts.

An additional point to clarify is that the GAIA dataset exhibits relatively low accuracy. This is primarily because GAIA tasks often require complex task planning and the use of multiple tools, posing significant challenges for all evaluated frameworks. It is important to note that the primary focus of this study is not on accuracy, but rather on comparing the performance overhead (e.g., time, token usage) across different frameworks. Therefore, we ensured that the accuracy across frameworks remains broadly comparable, without conducting detailed task-level progress analysis as seen in some related work. By carefully controlling experimental parameters, the fairness of our comparisons remains valid, even in the presence of lower absolute accuracy.

A.2 Detailed Evaluation Results

Table 4 5 6and 7 presents the detailed results obtained in this experiment. Unless stated otherwise, the times reported in the table are in seconds per query. The missing data corresponds to instances where the LLM failed to invoke the required tool correctly during the experiment—for example, by not returning outputs in the expected format or by not selecting the appropriate tool for invocation.The following are some noteworthy observations.

Insight 6: Token consumption may vary across frameworks even when executing the same workflow, owing to differences in implementation strategies.

In the results of ReAct workflow, it can be observed that even when using the same ReAct workflow, AgentScope exhibits a significant discrepancy in token usage between the GAIA and HumanEval datasets, with exceptionally high token consumption on GAIA. This is primarily because AgentScope includes the entire memory of the agent in the prompt during every LLM invocation. As the number of reasoning steps increases, the prompt length grows rapidly. While this issue is less apparent in the relatively simple HumanEval dataset, it becomes prominent in the more complex GAIA tasks.

Table 4: GAIA Detailed Results

	token			time					
frameworks	**prompt**	**output**	**total**	**llm**	**web_tool**	**pdf_tool**	**csv_tool**	**xlsx_tool**
LangChain	9358.35	637.92	9996.27	29.491	1.58856	0.02423455	0.00003333	0.06422606
AutoGen	1159.48	180.66	1340.15	8.464	9.4219	0.0009297	0.000336	0.002387
AgentScope	23520.479	785.891	24306.37	41.17	7.291	0.217	0.000297	0.00405
CrewAI	33621.857	664.511	34286.369	67.68	4.031	0.00965	0.000196	0.00422
LlamaIndex	20935.364	304.976	21240.339	27.244	1.4399	0.0001352	0.00016616	0.004254
Phidata	6386.667	323.558	6710.224	14.375	1.83012	0.001147	0.0007207	0.003858
PydanticAI	14459.17	320.588	14779.758	23.779	1.2275	0.001395	0.0003148	0.003795

time							
txt_tool	**docx_tool**	**audio_tool**	**vision_tool**	**video_tool**	**python_tool**	**total tool time**	**total time**
0.0004194	0.00009758	-	0.5345976	-	0.0152988	2.22746732	32.492
0.00002909	0.0002212	-	1.05489	-	0.00005333	10.4807	20.76
0.0000193	0.00000883	0.729	4.083	0.0000271	0.752	13.076	55.092
0.00123	0.000278	0.000346	0.03164	0.000999	0.09565	4.18	72.195
0.000034839	0.0001135	0.03341	0.8767	-	0.05782	2.4126	29.795
0.0002107	0.000073355	0.03821	1.4065	1.38445E-05	0.003035	3.2839	20.396
8.6865E-06	0.000056241	0.02965	1.2104	3.1952E-06	0.0001414	2.4732	26.238

Table 5: HumanEval Detailed Results

	token			time		
Framework	**prompt**	**output**	**total**	**llm**	**code executor**	**total**
LangChain	6326.36	617.13	6943.49	23.221	0.0034	23.968
AutoGen	767.45	106.34	873.79	5.822	0.0002	5.846
AgentScope	3180.689	561.518	3742.207	11.738	0.131	11.906
CrewAI	10817.65	892.798	11710.45	24.22	0.0258	25.24
LlamaIndex	1985.6	342.793	2328.152	9.52	0.003069	9.611
Phidata	967.329	354.427	1321.756	7.181	-	9.692
PydanticAI	812.951	352.543	1165.494	5.258	0.000007158	5.276

The high token usage observed in CrewAI's ReAct workflow can be attributed to the same reason. In fact, this issue is even more pronounced in CrewAI than in AgentScope, with significantly elevated token consumption observed across both the GAIA and HumanEval datasets.

In contrast, the majority of token consumption in LlamaIndex and Pydantic arises from the observation segments returned to the LLM after tool invocations. In the GAIA dataset, where tasks are complex and involve frequent tool usage, this results in substantial prompt token overhead.

There are also some issues observed in the MoA workflow. For example, PydanticAI does not require the invocation of all sub-agents during MoA execution, thereby reducing token consumption and runtime overhead.

Another example is that in the CrewAI framework, MoA is centrally managed by a global agent, which also plays the role of aggregation agent. The global agent receives the task and sequentially assigns it to sub-agents (e.g., agent1, agent2, agent3). Each sub-agent completes its part and returns the result to the global agent, which then decides the next step. After all agents have responded, the global agent summarizes the results and outputs the final answer.

In this setup, the global agent calls the LLM multiple times—once after each sub-agent's response. Because LLMs retain the full context of previous inputs and outputs in a single session, each new call includes all prior interactions. This leads to token accumulation, especially by the third or fourth step, where the prompt becomes much longer. As a result, total token usage becomes higher than in frameworks with different coordination or memory strategies.This phenomenon will become more apparent in Scalability part as the number of sub agents increases.

Insight 7: Parallel invocation reduces overall runtime.

Table 6: MMLU Detailed Results

Framework	token			time			
	prompt	**output**	**total**	**llm**	**embedding**	**retrieve**	**total**
LangChain	701.514	4.035	705.55	1.677	11.833	0.055	1.79
AutoGen	679.788	3.956	683.744	2.171	6.526	0.015	2.182
AgentScope	2664.315	2.878	2667.193	3.893	92.472	0.935	4.931
CrewAI	884.536	13.189	897.724	2.51	7.718	0.14	5
LlamaIndex	2079.702	50.339	2130.042	3.125	4.931	0.4303	3.575
Phidata	2797.441	37.347	2834.788	7.849	341.611	6.708	17.014
PydanticAI	6996.242	170.135	7166.378	9.685	5.977	0.03454	9.824

Table 7: AlpacaEval Detailed Results

Framework	token								
	llama			**qwen**			**deepseek**		
	prompt	output	total	prompt	output	total	prompt	output	total
LangChain	70.49	428.55	499.04	64.84	446.05	510.91	38.5	501.11	539.61
AutoGen	70.49	431.96	502.45	64.85	447.45	512.31	38.5	503.37	541.87
AgentScope	85.451	382.45	467.901	61.815	311.109	372.924	52.478	416.639	469.117
CrewAI	298.25	518.95	817.201	258.083	398.618	656.702	313.01	571.79	884.808
LlamaIndex	70.49	430.216	500.707	64.81	441.738	506.548	38.485	495.306	533.791
Phidata	118.846	438.078	556.924	93.899	463.795	557.694	83.391	440.691	524.082
PydanticAI	61.347	429.543	490.889	41.217	433.739	474.957	31.802	434.81	485.612

Framework	**gpt**			time					
	prompt	output	total	**llama**	**qwen**	**deepseek**	**aggregator**	**total**	agent1(llama)
LangChain	1522.48	444.81	1967.29	8.275	4.48	23.084	10.699	36.502	165.07/0
AutoGen	1529.96	450.63	1980.59	7.812	3.977	26.745	8.274	36.854	209.08/44.01
AgentScope	1138.243	352.564	1490.807	6.063	3.415	13.726	8.89	32.119	284.078/118
CrewAI	11694.576	679.15	12373.72	8.835	3.837	21.946	23.114	64	514.962/0
LlamaIndex	42.083	350.386	392.47	6.069	4.787	20.829	5.849	27.318	1180.078/898
Phidata	3003.319	756.689	3760.009	6.152	4.707	16.456	14.208	50.217	354.508/0
PydanticAI	1845.724	596.876	2442.6	6.503	3.441	17.79	27.486	46.45	96.022/0

Communication Size (content / wrapper bytes)

Framework	**prompt to agent**		**agent to aggregator**		
	agent2(qwen)	agent3(deepseek)	agent1(llama)	agent2(qwen)	agent3(deepseek)
LangChain	165.07/0	165.07/0	1983.02/3	2011.83/3	2072.98/3
AutoGen	209.08/44.01	209.08/44.01	2066.04/52.24	2071.24/57.38	2156.04/66.81
AgentScope	284.078/118	284.078/118	1659.318/124	1511.311/122	1889.247/126
CrewAI	483.740/0	619.516/0	2497.929/0	1754.701/0	2151.097/0
LlamaIndex	1171.078/889	1164.078/882	2022.417/33.689	2054.878/39.118	2116.377/48.641
Phidata	341.160/0	343.219/0	6128.259/2639.113	6131.272/2629.426	5715.126/2465.817
PydanticAI	95.425/0	97.116/0	2000.542/0	1927.093/0	1892.344/0

In some frameworks such as PydanticAI, the total runtime is observed to be shorter than the sum of individual tool and LLM invocation times on datasets such as GAIA and MoA. This improvement is attributed to its parallel execution architecture and asynchronous scheduling, which enables simultaneous invocation of multiple tools or LLMs, thereby effectively reducing end-to-end latency.

A.3 Scalability

To evaluate the scalability of the MoA workflow, we increased the number of worker agents from 3 to 6, 9, 12, and 15, while keeping the newly added agents identical in configuration to the original ones. Metrics from agents using the same LLM were aggregated for reporting. To clearly illustrate how efficiency evolves with increasing numbers of worker agents, we list separate tables (Table 8, 9, 10, 11, 12, 13, 14) for each framework.

Table 8: Scalability Evaluation of AlpacaEval Using AgentScope

Number of Worker Agent			3	6	9	12	15
Token	llama	prompt	85.451	137.84	206.76	275.68	344.6
		output	382.45	796.68	1204.91	1641.18	2021.9
		total	467.901	934.52	1411.67	1916.86	2366.5
	qwen	prompt	61.815	89.92	134.88	179.84	224.8
		output	311.109	555.47	848.94	1139.47	1497.77
		total	372.924	645.39	983.82	1319.31	1722.57
	deepseek	prompt	52.478	71.74	107.61	143.48	179.35
		output	416.639	841.37	1253.25	1704.54	2100.42
		total	469.117	913.11	1360.86	1848.02	2279.77
	gpt	prompt	1138.243	2237.83	3351.55	4542.02	5677.57
		output	352.564	412.43	439.44	442.62	434.15
		total	1490.807	2650.26	3790.99	4984.64	6111.72
Time	llama		6.063	12.76	19.307	25.547	35.311
	qwen		3.415	6.523	10.819	13.866	18.237
	deepseek		13.726	32.81	48.833	67.114	84.318
	gpt		8.89	15.468	14.33	14.373	15.813
	total		32.119	67.607	93.357	122.987	153.784
Communication	prompt to agent1		284.078/118	389.8/236	584.7/354	779.6/472	974.5/590
	prompt to agent2		284.078/118	389.8/236	584.7/354	779.6/472	974.5/590
	prompt to agent3		284.078/118	389.8/236	584.7/354	779.6/472	974.5/590
	agent1 to aggregator		1659.318/124	3256.270/250	4960.120/375	6718.820/500	8266.330/625
	agent2 to aggregator		1511.311/122	2375.700/246	4051.120/369	5477.260/492	7080.860/615
	agent3 to aggregator		1889.247/126	3705.450/254	5510.530/381	7497.600/508	9255.700/635

For some frameworks such as AgentScope, efficiency metrics exhibit near-linear growth as the number of worker agents increases, which aligns well with intuitive expectations.

Table 9: Scalability Evaluation of AlpacaEval Using LangChain

Number of Worker Agent			3	6	9	12	15
Token	llama	prompt	70.49	105.84	158.76	211.68	264.60
		output	428.55	1054.54	1518.52	2037.28	2537.08
		total	499.04	1160.38	1677.28	2248.96	2801.68
	qwen	prompt	64.84	93.92	140.88	187.84	234.80
		output	446.05	1007.95	1446.68	2017.43	2436.53
		total	510.91	1101.87	1587.56	2205.27	2671.33
	deepseek	prompt	38.50	41.74	62.61	83.48	104.35
		output	501.11	1132.22	1677.97	2224.98	2792.75
		total	539.61	1173.96	1740.58	2308.46	2897.10
	gpt	prompt	1522.48	3300.82	4734.44	6353.31	7823.07
		output	444.81	693.66	661.37	685.78	700.94
		total	1967.29	3994.48	5395.81	7039.09	8524.01
Time	llama		8.275	12.061	19.123	23.213	34.437
	qwen		4.480	10.838	16.584	24.812	29.335
	deepseek		23.084	40.801	66.156	73.476	115.888
	gpt		10.699	13.741	17.592	33.688	32.068
	total		36.502	37.958	47.112	59.725	66.075
Communication	prompt to agent1		165.07/0	153.76/0	230.64/0	307.52/0	384.4/0
	prompt to agent2		165.07/0	153.76/0	230.64/0	307.52/0	384.4/0
	prompt to agent3		165.07/0	153.76/0	230.64/0	307.52/0	384.4/0
	agent1 to aggregator		1983.02/3	4703.67/6	6787.84/9	9117.26/13	11314.20/17
	agent2 to aggregator		2011.83/3	4334.61/6	6286.30/9	8621.19/13	10546.46/17
	agent3 to aggregator		2072.98/3	4529.70/6	6702.62/9	8880.47/13	11164.34/17

Table 10: Scalability Evaluation of AlpacaEval Using AutoGen

Number of Worker Agent			3	6	9	12	15
Token	llama	prompt	70.49	104.14	158.76	211.68	264.6
		output	431.96	1004.94	1526.56	2028.21	2529.62
		total	502.45	1109.08	1685.32	2239.89	2794.22
	qwen	prompt	64.85	93.18	140.88	187.84	234.8
		output	447.45	993.87	1532.12	1940.46	2419.98
		total	512.31	1087.05	1673	2128.3	2654.78
	deepseek	prompt	38.5	40.68	62.61	83.48	104.35
		output	503.37	1109.77	1686.42	2249.68	2802.48
		total	541.87	1150.45	1749.03	2333.16	2906.83
	gpt	prompt	1529.96	3194.7	4830	6290.22	7807.46
		output	450.63	670.29	716.41	700.94	722.88
		total	1980.59	3864.99	5546.41	6991.16	8530.34
Time	llama		7.812	14.667	25.424	34.833	37.816
	qwen		3.977	12.653	21.064	28.736	35.71
	deepseek		26.745	46.011	71.345	71.98	104.207
	gpt		8.274	19.816	22.41	30.398	17.817
	total		36.854	47.339	50.843	55.6	46.428
Communication	prompt to agent1		209.08/44.01	236.48/86.04	359.7/129.06	479.6/172.08	599.5/215.1
	prompt to agent2		209.08/44.01	236.48/86.04	359.7/129.06	479.6/172.08	599.5/215.1
	prompt to agent3		209.08/44.01	236.48/86.04	359.7/129.06	479.6/172.08	599.5/215.1
	agent1 to aggregator		2066.04/52.24	4618.13/103.41	7069.64/156.52	9297.61/208.1	11541.91/258.55
	agent2 to aggregator		2071.24/57.38	4450.9/112.37	6777.28/172.84	8661.68/217.75	10768.69/271.29
	agent3 to aggregator		2156.04/66.81	4604.89/128.62	7399.95/204.09	9384.64/258.11	11497.32/317.86

However, frameworks that adopt parallel LLM invocation (e.g., AutoGen) effectively mitigate the increase in total execution time.

Table 11: Scalability Evaluation of AlpacaEval Using PydanticAI

Number of Worker Agent			3	6	9	12	15
Token	llama	prompt	61.347	95.5	126.29	139.77	161.71
		output	429.543	938.08	1273.35	1327.71	1559.8
		total	490.889	1033.58	1399.64	1467.48	1721.51
	qwen	prompt	41.217	58.39	76.32	80.85	94.52
		output	433.739	939.44	1213.31	1257.55	1608.87
		total	474.957	997.83	1289.63	1338.4	1703.39
	deepseek	prompt	31.802	41.44	50.5	50.88	58
		output	434.81	931.31	1210.15	1150.95	1311.62
		total	485.612	972.75	1260.65	1201.83	1369.62
	gpt	prompt	1845.724	3531.53	4673.28	4739.51	5684.52
		output	596.876	636.99	633.62	637.09	691.85
		total	2442.6	4168.52	5306.9	5376.6	6376.37
Time	llama		6.503	15.15	16.68	19.71	21.15
	qwen		3.441	8.38	11.2	11.59	13.41
	deepseek		17.79	33.34	42.14	40.71	47.19
	gpt		27.486	22.05	90.94	91.35	41.02
	total		46.45	42.24	110.78	111.4	62.13
Communication	prompt to agent1		96.022/0	88.12/0	113.86/0	124.09/0	134.34/0
	prompt to agent2		95.425/0	93.84/0	118.13/0	119.19/0	131.11/0
	prompt to agent3		97.116/0	94.73/0	108.99/0	103.12/0	113.92/0
	agent1 to aggregator		2000.542/0	4154.19/0	5693.77/0	6003.71/0	6851.79/0
	agent2 to aggregator		1927.093/0	4002.04/0	5302.46/0	5314.6/0	6682.15/0
	agent3 to aggregator		1892.344/0	3773.26/0	4941.13/0	4729.39/0	5284.75/0

Meanwhile, in some frameworks such as PydanticAI, the growth in token usage and related metrics is slightly slower than the increase in the number of worker agents. As discussed in Insight 6, PydanticAI does not enforce the invocation of all worker agents when implementing the MoA workflow, which results in fewer agent calls as the number of workers scales up.

Table 12: Scalability Evaluation of AlpacaEval Using CrewAI

Number of Worker Agent			3	6	9	12	15
Token	llama	prompt	298.25	536.95	706.39	760.54	795.95
		output	518.95	1186.13	1495.89	1597.78	1741.84
		total	817.201	1723.09	2202.26	2358.31	2537.63
	qwen	prompt	258.083	432.87	565.05	571.23	589.12
		output	398.618	862.44	1123.05	1088.7	1119.49
		total	656.702	1309.12	1688.11	1650.93	1708.61
	deepseek	prompt	313.01	432.87	526	544.75	668.04
		output	571.79	1007.86	1147.04	1181.72	1436.84
		total	884.808	1440.73	1673.04	1726.48	2104.88
	gpt	prompt	11694.576	28948.53	49040.19	54145.65	72234.23
		output	679.15	1136.86	1320.35	1363.42	1614.66
		total	12373.72	30085.4	50360.55	55509.07	73848.9
Time	llama		8.835	20.9	32.04	44.25	27.61
	qwen		3.837	7.7	16.64	13.84	14.61
	deepseek		21.946	32.49	48.37	50.72	45.43
	gpt		23.114	53.26	101.92	102.374	159.36
	total		64	120.54	212.76	218.34	245.26
Communication	prompt to agent1		514.962/0	925.12/0	1425.23/0	1724.32/0	1963.23/0
	prompt to agent2		483.740/0	912.35/0	1252.74/0	1328/0	1456.32/0
	prompt to agent3		619.516/0	900.54.5/0	1386.75/0	1327.32/0	1587.73/0
	agent1 to aggregator		2497.929/0	5921.52/0	7929.36/0	8623.56/0	9765.36/0
	agent2 to aggregator		1754.701/0	4421.22/0	6342.21/0	7021.42/0	8126.57/0
	agent3 to aggregator		2151.097/0	4783.14/0	6433.52/0	6798.21/0	7998.67/0

Table 13: Scalability Evaluation of AlpacaEval Using LlamaIndex

Number of Worker Agent			3	6	9	12	15
Token	llama	prompt	70.49	105.84	158.76	211.68	264.6
		output	430.216	1007.91	1502.61	2012.48	2501.66
		total	500.707	1113.75	1661.37	2224.16	2766.26
	qwen	prompt	64.81	93.92	140.88	187.84	234.8
		output	441.738	972.25	1431.39	1914.73	2420.34
		total	506.548	1066.17	1572.27	2102.57	2655.14
	deepseek	prompt	38.485	41.74	62.61	83.48	104.35
		output	495.306	1107.88	1695.19	2216.87	2794.66
		total	533.791	1149.62	1757.8	2300.35	2899.01
	gpt	prompt	42.083	24.68	24.68	24.68	24.68
		output	350.386	515.31	541.38	539.22	528.1
		total	392.47	539.99	566.06	563.9	552.78
Time	llama		6.069	12.44	18.98	25.65	35.58
	qwen		4.787	10.69	14.77	22.23	27.49
	deepseek		20.829	41.18	61.97	81.83	93.12
	gpt		5.849	9.39	9.66	10.4	16.06
	total		27.318	36.87	43.85	53.77	67.23
Communication	prompt to agent1		1180.078/898	2181.8/1796.0	3272.7/2694.0	4363.6/3592.0	5454.5/4490.0
	prompt to agent2		1171.078/889	2163.8/1778.0	3245.7/2667.0	4327.6/3556.0	5409.5/4445.0
	prompt to agent3		1164.078/882	2149.8/1764.0	3224.7/2646.0	4299.6/3528.0	5374.5/4410.0
	agent1 to aggregator		2022.417/33.689	4585.09/67.1	6813.56/99.75	9126.32/133.64	11342.05/169.22
	agent2 to aggregator		2054.878/39.118	4372.32/72.31	6456.6/106.13	8647.86/143.64	10907.85/181.15
	agent3 to aggregator		2116.377/48.641	4512.6/90.07	6923.71/137.8	9081.69/180.66	11437.99/227.25

A.4 Reproducibility Verification

To verify the reliability and reproducibility of our results, we conducted repeated experiments on the HumanEval dataset. The outcomes are reported in Table 5, 15, 16.

As illustrated by the error bars in Figure 5, the token data in our experiment is relatively stable. In contrast, the time data exhibits some variability due to network instability affecting LLM API response times. Nevertheless, the overall trend remains reproducible.

Table 14: Scalability Evaluation of AlpacaEval Using Phidata

Number of Worker Agent			3	6	9	12	15
Token	llama	prompt	118.846	114.5	110.58	116.61	118.06
		output	438.078	555.91	551.62	576.04	603.61
		total	556.924	670.41	662.2	692.65	721.67
	qwen	prompt	93.899	87.57	83.21	90.21	91.97
		output	463.795	634.08	621.29	663.48	707.2
		total	557.694	721.65	704.5	753.69	799.17
	deepseek	prompt	83.391	76.54	72.94	77.18	78.63
		output	440.691	505.74	525.82	527.8	545.07
		total	524.082	582.28	598.76	604.98	623.7
	gpt	prompt	3003.319	4180.34	5040.94	5973.25	6991.86
		output	756.689	785.45	778.76	795.1	801.86
		total	3760.009	4965.79	5819.7	6768.35	7793.72
Time	llama		6.152	6.55	6.55	9.12	10.33
	qwen		4.707	6.75	5.27	6.09	6.56
	deepseek		16.456	15.43	16.6	19.32	22.07
	gpt		14.208	23.13	25.68	31.67	31.7
	total		50.217	60.42	63.84	78.8	83.42
Communication	prompt to agent1		354.508/0	325.7/0	310.63/0	329.16/0	334.87/0
	prompt to agent2		341.160/0	309.01/0	293.84/0	319.17/0	326.58/0
	prompt to agent3		343.219/0	304.15/0	288.79/0	307.71/0	314.94/0
	agent1 to aggregator		6128.259/2639.113	7105.44/3163.16	6961.87/3105.45	7291.8/3252.42	7582.27/3388.25
	agent2 to aggregator		6131.272/2629.426	7475.54/3354.1	7269.53/3267.58	7792.87/3505.45	8121.06/3656.93
	agent3 to aggregator		5715.126/2465.817	6165.11/2699.97	6196.23/2734.51	6342.37/2791.33	6571.72/2891.08

Table 15: HumanEval Detailed Results 2

	token			**time**		
Framework	**prompt**	**output**	**total**	**llm**	**code executor**	**total**
LangChain	6769.16	695.15	7464.31	27.063	0.01267	27.82
AutoGen	790.29	108.26	898.55	5.685	0.000353	5.711
AgentScope	2429.72	530.323	2960.043	13.42	0.121	13.57
CrewAI	10026.98	914.96	10941.95	29.75	0.0432	30.47
LlamaIndex	2052	347.9	2399.9	19.81	0.00381	19.84
Phidata	1083.32	376.46	1459.79	11	8.99E-05	16.3
PydanticAI	903.6	353.48	1257.08	9.13	2.32E-05	9.15

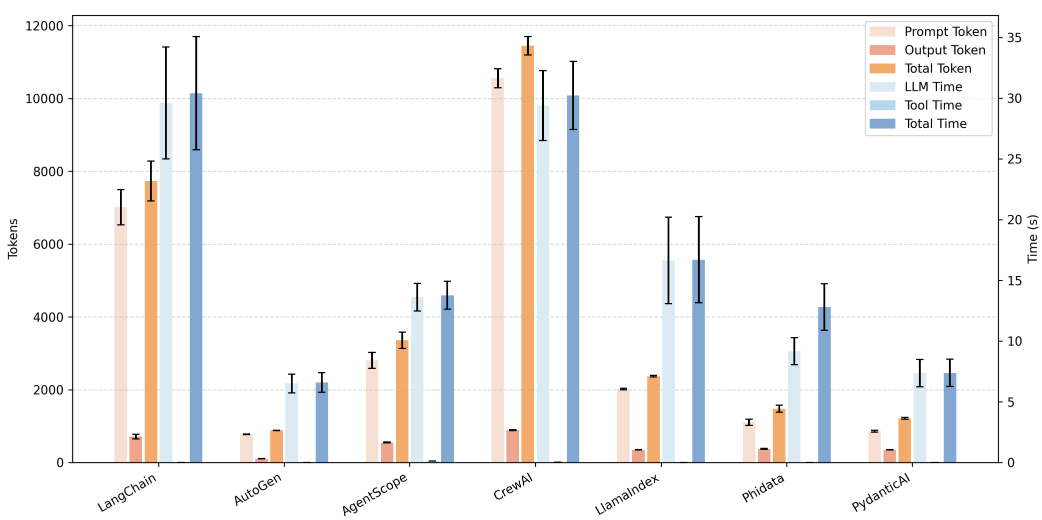


Figure 5: Consistency of Token Consumption and Latency in Repeated Experiments

Table 16: HumanEval Detailed Results 3

Framework	token			time		
	prompt	output	total	llm	code executor	total
LangChain	7953.34	832.63	8785.97	38.562	0.015723	39.471
AutoGen	769.72	105.78	875.5	8.027	0.000279	8.199
AgentScope	2804.341	568.36	3372.701	15.686	0.139	15.858
CrewAI	10822.16	867.08	11689.24	34.19	0.0342	34.98
LlamaIndex	2017.37	362.85	2380.23	20.61	0.00293	20.64
Phidata	1258.7	393.46	1652.16	9.36	0.000227	12.4
PydanticAI	874.49	340.66	1215.15	7.73	2.44E-05	7.74

A.5 Limitations and Broader Impacts

In this study, API calls were made exclusively by the LLM and Google Search tools. Due to potential network instability, the duration of these calls exhibited some degree of variability and randomness.

Moreover, most frameworks offer a wide range of tunable parameters. In our experiments, we adopted a simplified and uniform configuration across all frameworks for comparability, rather than tuning each individually for optimal performance. As such, the reported results may not reflect the upper-bound capabilities of each framework.

The potential positive societal impact of our benchmark lies in its ability to advance the development of more efficient and scalable AI agents. These improvements can help reduce computational costs, lower energy consumption, and enhance the feasibility of deploying AI systems in real-world applications such as education, healthcare, and assistive technologies.

At the same time, we recognize the possibility of indirect negative societal impacts. For example, broader availability and benchmarking of agent frameworks may inadvertently accelerate the deployment of autonomous systems without adequate oversight, or facilitate the misuse of agent-based automation in deceptive or manipulative contexts. However, AgentRace's primary goal is to promote efficiency in the evaluation of existing agent architectures and we believe that the societal benefits of a robust benchmarking infrastructure outweigh these risks—particularly when coupled with responsible deployment practices and clear usage guidelines.

B Experiment Details

In all experiments, the temperature was set to 0, the top k to 1 (if available), and all other parameters were set to their default values unless otherwise specified.

Except for the cases explicitly noted below, all workflows employ the default prompts provided by their respective frameworks, and the datasets are used without any modification to the original queries.

B.1 ReAct

For frameworks that do not have a specific implementation of ReAct, we use the following prompt to build the ReAct workflow:

```
You are a ReAct-based assistant.
You analyze the question, decide whether to call a tool or directly answer, and then
 respond accordingly.
Use the following format:Question: the input question or request
Thought: you should always think about what to do\nAction: the action to take (if
any)
Action Input: the input to the action (e.g., search query)
Observation: the result of the action
... (this process can repeat multiple times)
Thought: I now know the final answer
```

```
Final Answer: the final answer to the original input question or request
Begin!
Question: {input}
```

B.1.1 LangChain

Within the ReAct workflow implemented via LangChain's AgentExecutor, we set the max_iterations parameter to 15 for experiments on the GAIA dataset and to 10 for those on the HumanEval dataset.

B.2 RAG

For the following frameworks, we applied specific prompts to improve their token efficiency or to better align with the RAG workflow.

B.2.1 AutoGen

```
You are a helpful assistant. You can answer questions and provide information based
on the context provided.
```

B.2.2 CrewAI

```
You are a specialized agent for RAG tasks.You just need to give the answer of the
question. Don't need any othter word.Such as the answer is a number 5 ,you need
output '5'.Or the answer is A,you need to output 'A'.
```

B.2.3 Phidata

```
You are a RAG-based assistant. You analyze the question, and call the
search_knowledge_base tool to retrieve relevant documents from the knowledge base,
and then respond accordingly.
```

B.2.4 PydanticAI

```
You're a RAG agent. please search information from the given task to build a
knowledge base and then retrieve relevant information from the knowledge base.
```

B.3 MoA

Unless otherwise specified, the following prompt is used for the aggregator agent.

B.3.1 LangChain

```
You have been provided with a set of responses from various open-source models to
the latest user query. Your task is to synthesize these responses into a single,
high-quality response. It is crucial to critically evaluate the information provided
 in these responses, recognizing that some of it may be biased or incorrect. Your
response should not simply replicate the given answers but should offer a refined,
accurate, and comprehensive reply to the instruction. Ensure your response is well-
structured, coherent, and adheres to the highest standards of accuracy and
reliability.
```

B.3.2 AgentScope

```
You are an assistant called Dave,you should synthesize the answers from Alice, Bob
and Charles to arrive at the final response.
```

For the worker agent, we used the following prompt.

```
You are an assistant called Alice/Bob/Charles.
```

B.3.3 CrewAI

```
You are an agent manager, and You need to assign the questions you receive to each
of your all agents, and summarize their answers to get a more complete answer
You must give question to all the all agents, and you must summarize their answers
to get a more complete answer.\nYou need to be the best
```

For the worker agent, we used the following prompt.

```
You are one of the agents, you have to make your answers as perfect as possible,
there will be a management agent to choose the most perfect answer among the three
agents as output, you have to do your best to be selected
```

B.3.4 Phidata

```
Transfer task to all chat agents (There are 3 agents in your team)", "Aggreagate
responses from all chat agents
```

B.3.5 PydanticAI

```
Your task is to aggregate all agents results to solve complex tasks.\nYou analyze
the input, input the task to all tools that can run a single agent, and synthesize
the results from all agents into a final response.
```

B.4 GAIA

In this experiment, we used all levels of questions from the test subset of the GAIA dataset. Below are examples of prompts used in our system, depending on whether a file is attached:

```
question: A paper about AI regulation originally submitted to arXiv.org in June 2022
 features a figure with three axes, each labeled with a pair of opposing terms.
Which of these terms is used to describe a type of society in a Physics and Society
article submitted to arXiv.org on August 11, 2016?
```

```
question: The attached spreadsheet contains the inventory of a movie and video game
rental store located in Seattle, Washington. What is the title of the oldest Blu-Ray
 listed in this spreadsheet? Return it exactly as it appears., file_name: 32102e3e-
d12a-4209-9163-7b3a104efe5d.xlsx, file_path: path/to/32102e3e-d12a-4209-9163-7
b3a104efe5d.xlsx
```

B.5 HumanEval

To avoid generating explanatory text or pseudo-code that hinders automated accuracy evaluation, we slightly modify the original HumanEval queries by adding minimal prompts.Below is an example of the prompt used for HumanEval problems:

```
from typing import List

def has_close_elements(numbers: List[float], threshold: float) -> bool:
    """ Check if in given list of numbers, are any two numbers closer to each other
than
    given threshold.
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
    False
    >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
```

```
    True
    """

# Complete the function. Only return code. No explanation, no comments, no markdown.
```

B.6 MMLU

For the MMLU dataset, we constructed the vector database used in the RAG workflow based on the development subset and evaluated the performance of each framework using the test subset. Given the large number of tasks in this dataset, we used only one-quarter of them in our experiments. Considering that tasks from the same domain tend to be spatially adjacent in the dataset, we selected one out of every four tasks in index order. This sampling strategy ensures broader domain coverage and maintains fairness in the evaluation.

Below is an example of the question in MMLU:

```
Question:Find the degree for the given field extension Q(sqrt(2), sqrt(3), sqrt(18))
 over Q.
A.0
B.4
C.2
D.6
Answer with A, B, C, or D only
```

B.7 AlpacaEval

In this experiment, we used the full set of tasks for the basic MoA experiments, and the first 100 tasks for extended experiments involving more agents. Below is an example of one such task.

C Tool Implementation

For frameworks that do not include the required tools, we adopted a unified implementation as follows.

C.1 Search

C.1.1 AutogGen

```
def google_search(query: str, num_results: int = 2, max_chars: int = 500) -> list:
# type: ignore[type-arg]
    import os
    import time
    import requests
    from bs4 import BeautifulSoup
    from dotenv import load_dotenv
    load_dotenv()
    google_api_key = os.environ['GOOGLE_KEY']
    search_engine_id = os.environ['GOOGLE_ENGINE']
    if not search_engine_id or not search_engine_id:
        raise ValueError("API key or Search Engine ID not found")
    url = "https://www.googleapis.com/customsearch/v1"
    params = {
        "key": google_api_key,
        "cx": search_engine_id,
        "q": query,
        "num": num_results
    }
    response = requests.get(url, params=params)  # type: ignore[arg-type]
    if response.status_code != 200:
        print(response.json())
        raise Exception(f"Error in API request: {response.status_code}")
```

```
    results = response.json().get("items", [])
    def get_page_content(url: str) -> str:
        try:
            response = requests.get(url, timeout=10)
            soup = BeautifulSoup(response.content, "html.parser")
            text = soup.get_text(separator=" ", strip=True)
            words = text.split()
            content = ""
            for word in words:
                if len(content) + len(word) + 1 > max_chars:
                    break
                content += " " + word
            return content.strip()
        except Exception as e:
            print(f"Error fetching {url}: {str(e)}")
            return ""
    enriched_results = []
    for item in results:
        body = get_page_content(item["link"])
        enriched_results.append(
            {
                "title": item["title"],
                "link": item["link"],
                "snippet": item["snippet"],
                "body": body
            }
        )
        time.sleep(1)
    return enriched_results
```

C.1.2 PydanticAI

```
def google_search(query, num=None):
    """
    Make a query to the Google search engine to receive a list of results.
    Args:
        query (str): The query to be passed to Google search.
        num (int, optional): The number of search results to return. Defaults to
None.

    Returns:
        str: The JSON response from the Google search API.

    Raises:
        ValueError: If the 'num' is not an integer between 1 and 10.
    """
    try:
        QUERY_URL_TMPL = ("https://www.googleapis.com/customsearch/v1?key={key}&cx={
engine}&q={query}")
        url = QUERY_URL_TMPL.format(
            key=os.environ['GOOGLE_KEY'],
            engine=os.environ['GOOGLE_ENGINE'],
            query=urllib.parse.quote_plus(str(query))
        )
        if num is not None:
            if not 1 <= num <= 10:
                raise ValueError("num should be an integer between 1 and 10,
inclusive")
            url += f"&num={num}"
        response = requests.get(url)
        return response.text
    except Exception as e:
        return f"Error: {e}"
```

C.2 PDF loader

```
def pdf_load(file_path: str) -> ServiceResponse:
    try:
        reader = PdfReader(file_path)
        text = ""
        for page in reader.pages:
            text += page.extract_text() + "\n"
        return ServiceResponse(status=ServiceExecStatus.SUCCESS,content=text)
    except Exception as e:
        return ServiceResponse(ServiceExecStatus.ERROR, str(e))
```

C.3 CSV reader

```
import pandas as pd

def csv_load(path:str)->ServiceResponse:
    try:
        df = pd.read_csv(path)
        csv_str = df.to_string(index=False)
        return ServiceResponse(status=ServiceExecStatus.SUCCESS,content=csv_str)
    except Exception as e:
        return ServiceResponse(ServiceExecStatus.ERROR, str(e))
```

C.4 XLSX reader

```
def xlsx_load(path:str)->ServiceResponse:
    try:
        excel_file = pd.read_excel(path, sheet_name=None)
        result = ""
        for sheet_name, df in excel_file.items():
            result += f"Sheet: {sheet_name}\n"
            result += df.to_string(index=False) + "\n\n"
        return ServiceResponse(status=ServiceExecStatus.SUCCESS,content=result.strip
())
    except Exception as e:
        return ServiceResponse(ServiceExecStatus.ERROR, str(e))
```

C.5 Text file reader

```
import pandas as pd

def txt_load(path:str)->ServiceResponse:
    try:
        with open(path, 'r', encoding='utf-8') as f:
            txt_str = f.read()
        return ServiceResponse(status=ServiceExecStatus.SUCCESS,content=txt_str)
    except Exception as e:
        return ServiceResponse(ServiceExecStatus.ERROR, str(e))
```

C.6 Docx reader

```
from docx import Document

def docs_load(path:str)->ServiceResponse:
    try:
        doc = Document(path)
        docx_str = "\n".join([para.text for para in doc.paragraphs])
        return ServiceResponse(status=ServiceExecStatus.SUCCESS,content=docx_str)
```

```
    except Exception as e:
        return ServiceResponse(ServiceExecStatus.ERROR, str(e))
```

C.7 MP3 loader

```
import whisper
from typing import cast

def load_audio(file):
    model = whisper.load_model(name="base")
    model = cast(whisper.Whisper, model)
    result = model.transcribe(str(file))
    return result["text"]
```

C.8 Figure loader

```
from transformers import DonutProcessor, VisionEncoderDecoderModel
import re
from PIL import Image

def load_image(path):
    image = Image.open(path)
    processor = DonutProcessor.from_pretrained(
                        "naver-clova-ix/donut-base-finetuned-cord-v2"
                    )
    model = VisionEncoderDecoderModel.from_pretrained(
        "naver-clova-ix/donut-base-finetuned-cord-v2"
    )
    device = 'cpu'
    model.to(device)
    # prepare decoder inputs
    task_prompt = "<s_cord-v2>"
    decoder_input_ids = processor.tokenizer(
        task_prompt, add_special_tokens=False, return_tensors="pt"
    ).input_ids
    pixel_values = processor(image, return_tensors="pt").pixel_values
    outputs = model.generate(
        pixel_values.to(device),
        decoder_input_ids=decoder_input_ids.to(device),
        max_length=model.decoder.config.max_position_embeddings,
        early_stopping=True,
        pad_token_id=processor.tokenizer.pad_token_id,
        eos_token_id=processor.tokenizer.eos_token_id,
        use_cache=True,
        num_beams=3,
        bad_words_ids=[[processor.tokenizer.unk_token_id]],
        return_dict_in_generate=True,
    )
    sequence = processor.batch_decode(outputs.sequences)[0]
    sequence = sequence.replace(processor.tokenizer.eos_token, "").replace(
        processor.tokenizer.pad_token, ""
    )
    # remove first task start token
    text_str = re.sub(r"<.*?>", "", sequence, count=1).strip()
    return text_str
```

C.9 Video loader

```
import whisper
from typing import cast
from pydub import AudioSegment
```

```
from pathlib import Path

def load_video(file):
    video = AudioSegment.from_file(Path(file), format=file[-3:])
    audio = video.split_to_mono()[0]
    file_str = str(file)[:-4] + ".mp3"
    audio.export(file_str, format="mp3")
    model = whisper.load_model(name="base")
    model = cast(whisper.Whisper, model)
    result = model.transcribe(str(file))
    return result["text"]
```

C.10 data retrieval

```
    def create_vector_db():
    import faiss
    import pickle
    from sentence_transformers import SentenceTransformer
    from data.mmlu import merge_csv_files_in_folder
    dataset=merge_csv_files_in_folder(path to MMLU/dev)
    docs = []
    for item in dataset:
            text = item[0].replace(",please answer A,B,C,or D.",",")+f"answer:{item
[1]}."
            docs.append(text)
    embed_model = SentenceTransformer('all-MiniLM-L6-v2')
    doc_embeddings = embed_model.encode(docs)
    dimension = doc_embeddings.shape[1]
    index = faiss.IndexFlatL2(dimension)
    index.add(doc_embeddings)
    faiss.write_index(index, "db/index.faiss")
    with open("db/index.pkl", "wb") as f:
        pickle.dump(docs, f)

def load_vector_db():
    import faiss
    import pickle
    from sentence_transformers import SentenceTransformer
    class db:
        def __init__(self):
            self.index = faiss.read_index("db/index.faiss")
            with open("db/index.pkl", "rb") as f:
                self.docs = pickle.load(f)
            self.embed_model = SentenceTransformer('all-MiniLM-L6-v2')
        def search(self, query, k=5):
            query_embedding = self.embed_model.encode([query])
            D, I = self.index.search(query_embedding, k)
            return [self.docs[i] for i in I[0]]
    return db()
```

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction provide a clear overview of the benchmark (AgentRace), its focus (efficiency), and the scope of evaluation (7 frameworks, 3 workflows, 4 datasets), matching the content presented in Sections 3, 4, 5, 6.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations, such as the instability of LLM and search times due to network issues, are discussed in the Appendix.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The paper emphasizes experimental evaluation and does not include theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper details the experimental setup, datasets, and evaluation metrics in Section 5. Additional testing details are provided in the Appendix. The code is also open-sourced.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The implementation is publicly available on GitHub, and the four evaluation datasets (GAIA, HumanEval, MMLU, AlpacaEval) are open-source and accessible via Hugging Face.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental settings, including datasets, models, and other evaluation details such as hyperparameters, are described in Section 5 and the Appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The paper reports experimental results based on a consistent setup, with statistical tests included in the Appendix. These help convey the stability and significance of the findings.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper specifies the hardware used (12-core Intel(R) Xeon(R) Silver 4214R CPUs and a single NVIDIA RTX 3080 Ti GPU) as well as the execution time in Section 5.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research uses publicly available datasets and models and it complies with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Please refer to the Appendix of supplementary material.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: The paper proposes a new benchmark, which is unlikely to pose any substantial risks of misuse.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All datasets and models used in this paper are publicly available and appropriately cited. We have ensured that their usage complies with the licenses and terms provided by the original creators.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Please refer to the supplementary material.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing experiments or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.