Peeking Under the Hood of Multi-Agent Systems

Tie Ma*
Beihang University
Beijing, China
tie.ma@kaust.edu.sa

Yixi Chen*
KAUST
Thuwal, Saudi Arabia
yixi.chen@kaust.edu.sa

Vaastav Anand
MPI-SWS
Saarbrücken, Germany
vaastav@mpi-sws.org

Alessandro Cornacchia
KAUST
Thuwal, Saudi Arabia
alessandro.cornacchia@kaust.edu.sa

Amândio R. Faustino
KAUST
Thuwal, Saudi Arabia
amandio.faustino@kaust.edu.sa

Guanheng Liu
Beihang University
Beijing, China
liuguanheng@buaa.edu.cn

Shan Zhang
Beihang University
Beijing, China
zhangshan18@buaa.edu.cn

Hongbin Luo
Beihang University
Beijing, China
luohb@buaa.edu.cn

Suhaib A. Fahmy
KAUST
Thuwal, Saudi Arabia
suhaib.fahmy@kaust.edu.sa

Zafar A. Qazi
KAUST
Thuwal, Saudi Arabia
LUMS
Lahore, Pakistan
zafar.qazi@kaust.edu.sa

Marco Canini
KAUST
Thuwal, Saudi Arabia
marco@kaust.edu.sa

Abstract

Multi-agent systems (MASes) powered by large language models (LLMs) are increasingly deployed in real applications, yet practitioners still lack a practical way to systematically compare and tune key system choices such as backend LLMs, agent frameworks, and MAS architectures. The stochastic and failure-prone nature of runtime LLM decisions further complicates controlled experimentation and ablation. While existing benchmarks emphasize application-level outcomes (e.g., task success), they provide limited support for studying how system knobs influence end-to-end behavior. Similarly, current telemetry tools capture raw execution data but offer no standard for identifying which metrics actually impact system reliability.

We introduce MAESTRO, an open-source platform that defines and standardizes actionable evaluation metrics for MAS deployment. By profiling 12 representative MASes, MAESTRO distills complex execution dynamics into critical diagnostic metrics. Demonstrating this framework, our evaluation reveals that while high-level interaction structures remain stable, execution order varies substantially, and tool usage exhibits strong architecture-dependent cost-accuracy trade-offs. Ultimately, MAESTRO transitions MAS evaluation toward targeted, metric-driven optimization.

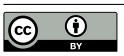
CCS Concepts

• **Computing methodologies** → **Multi-agent systems**; • **General and reference** → **Measurement**.

Keywords

Multi-agent systems, Large language models, Benchmarking, Observability, System reliability, Performance evaluation

*Equal contribution. Tie Ma's work was done during an internship at KAUST.



This work is licensed under a Creative Commons Attribution 4.0 International License.
CAIS '26, San Jose, CA, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2415-2/26/05
https://doi.org/10.1145/3786335.3813204

ACM Reference Format:
Tie Ma, Yixi Chen, Vaastav Anand, Alessandro Cornacchia, Amândio R. Faustino, Guanheng Liu, Shan Zhang, Hongbin Luo, Suhaib A. Fahmy, Zafar A. Qazi, and Marco Canini. 2026. Peeking Under the Hood of Multi-Agent Systems. In *ACM Conference on AI and Agentic Systems (CAIS '26), May 26–29, 2026, San Jose, CA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3786335.3813204

1 Introduction

Large language model (LLM)-based multi-agent systems (MASes) are increasingly deployed as a powerful paradigm for complex applications. They have demonstrated utility across diverse domains, ranging from cooperative software development [2] and embodied AI [14] to complex financial [15] and social simulations [6].

However, LLM-based multi-agent systems introduce new challenges due to their stochastic execution and reliance on runtime LLM outputs, resulting in highly non-deterministic behavior that limits the applicability of optimization techniques designed for deterministic workflows [16]. Furthermore, the diversity in MAS architectures (e.g., plan-and-execute [12], divide-and-conquer [18]) and communication protocols dictates that the relevance of operational metrics shifts dramatically depending on the system design. A diagnostic signal critical for one interaction pattern may be irrelevant noise in another, complicating classical trade-offs between resource efficiency and reliability. Together, these factors break common assumptions of predictability and observability, demanding a benchmarking approach that moves beyond raw telemetry collection to identify and standardize architecture-aware diagnostic metrics.

Existing evaluation methodologies fail to provide a comprehensive understanding of MAS execution behavior. While a large body of work focuses on evaluating LLM serving performance [3, 13], the rapid adoption of LLM-based agents has led to a growing set of MAS benchmarks [1, 4, 16, 19]. These benchmarks primarily evaluate individual agent capabilities while treating MASes as black boxes focused on end-to-end application-level outcomes. In practice, such evaluations indicate *whether* performance degrades, but fail to explain *why*.

Conversely, while general-purpose telemetry tools collect fine-grained execution data, they strictly profile individual runs. They offer no mechanism to aggregate stochastic traces and determine which metrics actually diagnose system health across different architectures. Consequently, MAS architects and system operators are trapped between benchmarks lacking system-level observability and telemetry tools lacking actionable, multi-run context. Consistent with this gap, a recent survey [9] reports that nearly 75% of teams deploying production MASes evaluate their systems without standardized benchmarks, relying instead on custom, in-house solutions.

To address this gap, we present a demo of MAESTRO, a Multi-Agent Evaluation Suite for Testing, Reliability, and Observability. Rather than focusing solely on isolated debugging tasks, MAESTRO serves as a comprehensive benchmarking and exploration framework built specifically for MAS architects and system operators. It provides a unified repository of 12 representative MASes alongside a declarative integration interface that allows users to seamlessly profile custom agent architectures without extensive code modification.

Crucially, MAESTRO serves as an engine for metric discovery, moving beyond ad-hoc observation to establish standardized diagnostic signals. For example, to decouple inherent LLM stochasticity from underlying structural bottlenecks, MAESTRO specifically tracks call-graph similarity. Using this metric, our analysis reveals that MAS call-graph structures remain structurally stable despite substantial temporal variability in execution sequences (*Finding 1* in § 3.1), exposing specific optimization opportunities like caching and prefetching. Furthermore, we demonstrate how MAESTRO exposes the architecture-dependent overhead of external tools, quantifying severe trade-offs between accuracy gains and resource costs (*Finding 2* in § 3.2). Collectively, the MAESTRO platform and these metric-driven evaluations provide a principled foundation for designing and deploying robust MASes. MAESTRO's code, dataset, and demo video are available at https://maestro-suite.github.io/.

2 MAESTRO design

We first detail MAESTRO's architecture and control flow, followed by the built-in MAS suites of MAESTRO. Finally, we outline a unified protocol standardizing inputs, outputs, and analysis to enable cross-MAS comparisons.

MAESTRO architecture. Figure 1 illustrates MAESTRO's linear workflow: users prepare and configure MAS instances, triggering runtime execution and trace collection, followed by post-hoc processing to derive interpretable results.

MAS preparation. MAESTRO provides 12 built-in MASes, enabling out-of-the-box evaluation without additional integration effort. To incorporate custom MASes or external frameworks, users define a declarative JSON configuration. By leveraging integration libraries like any-agent [7], MAESTRO interprets these configurations to instantiate agents across different frameworks and map custom tools. This reduces integration overhead to specifying coordination topologies (e.g., round-robin, handoff) and agent schemas, exposing new MAS implementations to MAESTRO's configuration and observation interface.

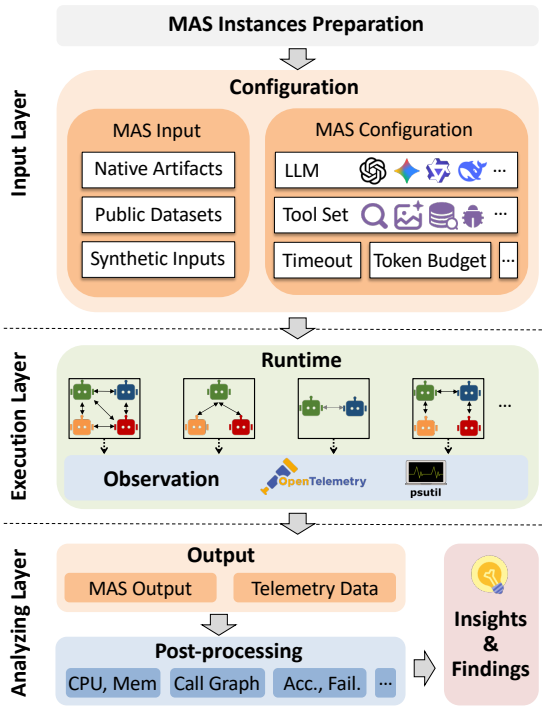


Figure 1: MAESTRO architecture overview.

Configuration. Based on the prepared MASes, users specify the evaluation setup, including input sources, LLM configuration of each agent instance, external tool access, etc.

Runtime. Based on the provided configuration, the Runtime component orchestrates the instantiation and execution of the MASes. Task inputs are continuously fed into the runtime, triggering agent interactions, tool invocations, and control-flow decisions.

Observation. During execution, the Observation component monitors system behavior through function-call hooks or sampling-based instrumentation. Built on top of OpenTelemetry [8] and Psutil [10], it records both execution metrics (e.g., latency, token usage) and additional signals specified in the configuration. Collected traces are forwarded to the Post-processing component.

Post-processing. The *Post-processing* component aggregates and analyzes execution traces to make MAS behavior inspectable (e.g., CPU, call graph). These summaries enable users to explore execution trajectories, compare configurations across runs, and identify performance bottlenecks and sources of instability.

Representative built-in MASes. We include 12 representative MASes from official repositories of major agentic frameworks as built-in MASes (Table 1) and group them into two evaluation suites: the **Full-suite (F)** contains all MASes, whereas the **Architecture-focused suite (A)** isolates architectural effects by comparing Corrective RAG (CR), Plan and Execute (PE), and LATS (LA) on the same tasks [17] under different interaction patterns. Figure 2 summarizes these three architectures: CR augments generation with corrective retrieval, PE performs explicit planning with iterative execution/re-planning, and LA refines solutions via tree-search. Detailed MAS selection criteria is provided in [5].

Table 1: Selected MASes overview. The "Suite" column indicates membership, F: Full-suite; A: Architecture-focused suite.

MAS Name	Abbr.	App. Domain	Framework	Interaction			Data Spec.		Suite
				Type	#Agt	#Tool	In	Out	
Financial Analyzer	FA	Finance	MCP-Agent	Correct	6	1	Artifacts	Opn-End	F
Image Scoring	IM	Creativity	ADK	Debate	4	2	Synthetic	Cls-Form	F
Marketing	MA	Marketing	ADK	Coordination	4	1	Synthetic	Opn-End	F
Brand Search Opt.	BO	Marketing	ADK	Coordination	4	10	Synthetic	Opn-End	F
Content Creation	CC	Creativity	ADK	Plan	4	1	Synthetic	Opn-End	F
Magentic-One	MO	Cross-domain	Autogen	Plan	4	0	Artifacts	Opn-End	F
Stock Research	SR	Finance	Autogen	Coordination	4	2	Artifacts	Opn-End	F
Travel Planning	TP	Travel	Autogen	Coordination	4	0	Artifacts	Opn-End	F
Tree of Thought	TT	Cross-domain	LangGraph	Debate	3	0	Artifacts	Cls-Form	F
Corrective RAG	CR	Cross-domain	LangGraph	Coordination	5	2	Datasets	Opn-End	F,A
Plan and Execution	PE	Cross-domain	LangGraph	Plan	3	1	Datasets	Opn-End	F,A
LATS	LA	Cross-domain	LangGraph	Plan	3	1	Datasets	Opn-End	F,A

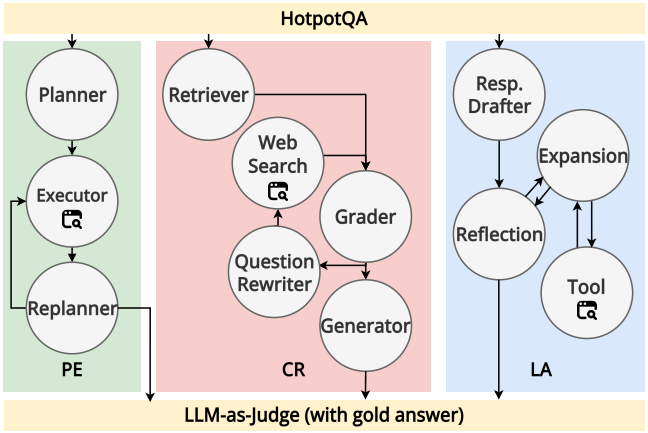


Figure 2: Three architectures solving a unified task.

Unified evaluation protocol. We standardize inputs, outputs, and analysis to enable fair comparisons across heterogeneous MAS implementations. Inputs are derived from (i) official artifacts, (ii) synthetic prompts, or (iii) public datasets; outputs follow two templates that capture detailed resource usage and structured execution traces; and evaluation aggregates system-, application-, and semantic-level metrics (with an LLM-as-judge when needed). Full protocol details are provided in [5].

3 Characterizing MASes with MAESTRO

To demonstrate MAESTRO's analytical capabilities, we report one representative experiment for each evaluation suite.

Setup. For each MAS, we conduct 20 independent runs, each with a single user-level input; runs are capped at 10 minutes to prevent infinite loops. Specifically, for the Architecture-focused suite, we cap each LLM response at 8,192 tokens. We use Tavily [11] as the web search tool when enabled. When correctness is required, we use GPT-4o-mini as an LLM-as-judge.[1] We use the following abbreviations throughout: Gemini-2.0-flash-lite (Ge20FL), Gemini-2.5-flash-lite (Ge25FL), Gemini-2.5-flash (Ge25F), GPT-4o-mini (G4oM), GPT-5-mini (G5M), and GPT-5-nano (G5N).

3.1 How stable are MAS call graphs, and what factors influence their variability?

Different from traditional software systems, LLM-based MASes exhibit inherent execution stochasticity, making system-level stability a prerequisite for reproducible evaluation. Because this stochasticity manifests independently in both the occurrence and the sequence of agent interactions, it is necessary to decouple structural stability from temporal variability. To capture this duality more comprehensively, we quantify run-to-run call-graph stability using two complementary metrics: Jaccard similarity (for order-agnostic interaction overlap) and LCS similarity (for order-aware trace alignment) [5].

MAS execution is structurally stable but temporally variable. Across repeated runs, call graphs exhibit high structural overlap (averaging a Jaccard similarity of 0.79) but moderate temporal alignment (averaging an LCS of 0.61), as illustrated in Figure 3. This discrepancy indicates that while MASes reliably invoke the same subset of agents to solve a task, the specific sequence of these interactions remains highly susceptible to runtime stochasticity. PE and SR perfectly exemplify this high-Jaccard, low-LCS paradigm; their macro-level interaction topologies are structurally stable, yet their LLM-driven planning and coordination phases introduce substantial variability in execution order. TT acts as a rigid exception, maintaining perfect stability (1.0 for both metrics) because Autogen's RoundRobinGroupChat enforces a deterministic execution sequence that strictly eliminates dynamic routing. Conversely, FA

[1] While we acknowledge the absence of human-calibrated inter-rater reliability and note that reasoning-heavy models (e.g., `o3` or `Claude 3.5 Sonnet`) could serve as stronger judges, `GPT-4o-mini` was selected to practically balance evaluation costs across the 240+ stochastic runs required to evaluate structural stability.

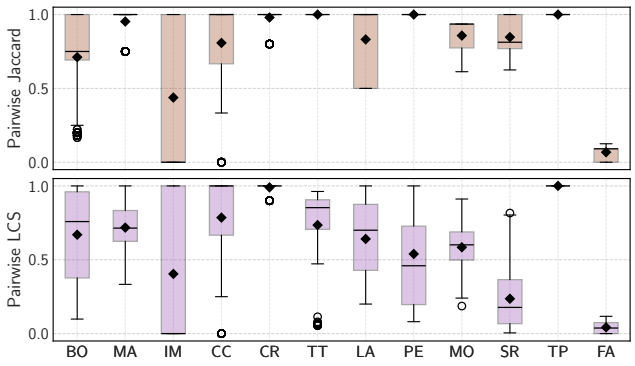


Figure 3: Cross-model call graph similarity.

emerges as a severe structural outlier (Jaccard 0.07, LCS 0.04) dictated by its MCP-AGENT-based dynamic orchestration. Its central orchestrator continuously adapts the topology at runtime, deliberately selecting or bypassing specific sub-agents, such as the *financial-analyst* or *report-writer* stages, based on intermediate data quality assessments and predefined early-stop conditions. This extreme statistical variance reflects deliberate, inherent architectural flexibility rather than arbitrary execution noise.

Finding 1: Across runs, MAS call graphs are largely stable in which agent-to-agent interactions occur, but often unstable in the order those interactions unfold.

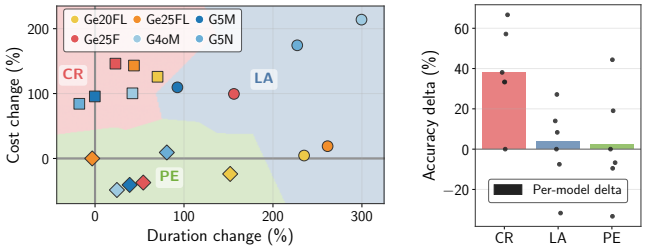
Guidelines. The observed structural stability of MAS interaction graphs enables more efficient caching and scheduling strategies that exploit repeated interaction structure. Meanwhile, temporal variability in execution order motivates using repeated runs and order-aware analysis when diagnosing behavior.

3.2 How does tool usage impact cost and accuracy?

Enabling external tools is often assumed to improve MAS quality by providing additional evidence or capabilities. We find that tool effects are strongly architecture-dependent, and can trade off accuracy against cost and latency.

Tool impact is non-uniform and architecture-dependent. Although enabling tools is widely assumed to improve MAS quality, we reveal that tool effects are strongly architecture-dependent. As shown in Figure 4a, integrating web search commonly increases overall execution overhead, but the magnitude and specific areas affected depend on the workflow integration. CR primarily incurs a higher monetary cost (median +$0.0010 per task) with only a modest impact on latency. In contrast, PE incurs a substantial latency penalty (median +34.1 s) while slightly reducing monetary cost, indicating a shift from token expenditure to longer execution times. LA exhibits the largest overhead overall, simultaneously increasing both cost and latency.

Despite these broader trends, providing external context can occasionally reduce execution overhead by curbing speculative generation. For instance, CR coupled with GPT-5-nano reduces mean



(a) Cost and latency deltas. **(b) Accuracy deltas.**

Figure 4: Impact of web search on cost and accuracy. Deltas are calculated as with − without search.

task duration by 13.9% because external evidence yields shorter, more confident outputs that successfully offset the initial retrieval overhead. Similarly, for PE, web search forces plans to become much more concrete and concise; average planner token consumption drops from over 1,500 to a few hundred per call, and subsequent planning steps shorten. Even when the total number of iterative planning cycles remains similar, the reduced per-step token usage easily outweighs the base retrieval cost.

Tool effects on accuracy vary substantially by architecture (Figure 4b). CR benefits consistently, achieving a large median accuracy gain (+35.7%) and improving in the majority of runs. Conversely, PE often fails to benefit and can actively degrade in accuracy, while LA demonstrates only marginal and unstable improvements. Ultimately, these accuracy trends directly mirror the corresponding cost and latency trade-offs: CR incurs modest overhead, whereas PE primarily increases latency, and LA increases both cost and latency.

Finding 2: External tools can reduce speculative generation and cost in specific contexts, but they only reliably improve accuracy when the agent architecture can integrate them without amplifying execution overhead or instability.

Guidelines. System design dictates the type of overhead introduced by external tools; operators must therefore treat the choice of execution model as a strategic trade-off between minimizing latency and optimizing token budget. Overall, tools help only when the architecture can integrate them without amplifying execution overhead or instability.

4 Conclusion and future work

In this demo paper, we presented MAESTRO, a comprehensive benchmarking framework that enables practitioners to systematically explore diverse setups for LLM-based MASes. Moving beyond black-box application outcomes, our demo reveals that MAS call graphs are structurally stable yet temporally variable, and that external tools trade latency against cost without guaranteeing accuracy improvement.

To enhance practitioner observability, immediate future work replaces the current CLI workflow with a lightweight web dashboard for comprehensive real-time visualization of call-graphs and trade-offs. Furthermore, future work will extend MAESTRO to distributed MAS architectures to evaluate how network latency, synchronization, and consistency impact scalability. We will analyze

how communication protocols (e.g., A2A, MCP) and parallel execution alter system dynamics, coordination overhead, and emergent failures.

Acknowledgments

We would like to thank the anonymous reviewers for their constructive comments and insightful feedback. This work was supported in part by the Nature Science Foundation of China under Grant 624B2015, 62422201, 62271019, 62225201, U24B20128, in part by the Fundamental Research Funds for the Central Universities, China, and State Key Laboratory of Complex & Critical Software Environment, in part by the China Scholarship Council under Grant No. 202506020109.

References

[1] Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. StableToolBench: Towards Stable Large-Scale Benchmarking on Tool Learning of Large Language Models. In *ACL*. Bangkok, Thailand, 11143–11156.

[2] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. 2023. MetaGPT: Meta programming for a multi-agent collaborative framework. In *ICLR*.

[3] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Alexander Cosgrove, Christopher D Manning, Christopher Re, Diana Acosta-Navas, Drew Arad Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue WANG, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri S. Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Andrew Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. 2023. Holistic evaluation of language models. *TMLR* (2023).

[4] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2024. AgentBench: Evaluating LLMs as Agents. In *ICLR*.

[5] Tie Ma, Yixi Chen, Vaastav Anand, Alessandro Cornacchia, Amândio R Faustino, Guanheng Liu, Shan Zhang, Hongbin Luo, Suhaib A Fahmy, Zafar A Qazi, et al. 2026. MAESTRO: Multi-Agent Evaluation Suite for Testing, Reliability, and Observability. *arXiv preprint arXiv:2601.00481* (2026).

[6] Ahmad Mouri Zadeh Khaki, Ahyoung Choi, and Laleh Seyyed-Kalantari. 2025. Simulating Social Behavior of LLM-Based Autonomous Negotiator Agents in a Game-Theoretical Framework Using Multi-Agent Systems. *International Journal of Human–Computer Interaction* (2025), 1–10.

[7] mozilla ai. 2026. *any-agent*. https://mozilla-ai.github.io/any-agent/ Accessed: 2026-02-11.

[8] OpenTelemetry. 2025. *OpenTelemetry*. https://opentelemetry.io/ Accessed: 2025-12-28.

[9] Melissa Z Pan, Negar Arabzadeh, Riccardo Cogo, Yuxuan Zhu, Alexander Xiong, Lakshya A Agrawal, Huanzhi Mao, Emma Shen, Sid Pallerla, Liana Patel, et al. 2025. Measuring agents in production. *arXiv preprint arXiv:2512.04123* (2025).

[10] Psutil. 2025. *Psutil: process and system utilities*. https://github.com/giampaolo/psutil Accessed: 2025-12-28.

[11] Tavily. 2026. *Tavily*. https://www.tavily.com/ Accessed: 2026-01-23.

[12] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*. Toronto, Canada, 2609–2634.

[13] Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Yuchu Fang, Yeju Zhou, Yang Zheng, Zhenheng Tang, Xin He, Rui Guo, et al. 2025. Burstgpt: A real-world workload dataset to optimize llm serving systems. In *SIGKDD*. 5831–5841.

[14] Di Wu, Xian Wei, Guang Chen, Hao Shen, and Bo Jin. 2025. Generative Multi-Agent Collaboration in Embodied AI: A Systematic Review. In *IJCAI*. 10723–10732. Survey Track.

[15] Yijia Xiao, Edward Sun, Di Luo, and Wei Wang. 2024. TradingAgents: Multi-agents LLM financial trading framework. *arXiv preprint arXiv:2412.20138* (2024).

[16] Bingyu Yan, Zhibo Zhou, Litian Zhang, Lian Zhang, Ziyi Zhou, Dezhuang Miao, Zhoujun Li, Chaozhuo Li, and Xiaoming Zhang. 2025. Beyond self-talk: a communication-centric survey of LLM-based multi-agent systems. *arXiv preprint arXiv:2502.14321* (2025).

[17] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *EMNLP*.

[18] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024. Language agent tree search unifies reasoning, acting, and planning in language models. In *ICML*.

[19] Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Daisy Zhe Wang, Zhenhailong Wang, Cheng Qian, Robert Tang, Heng Ji, et al. 2025. Multiagentbench: Evaluating the collaboration and competition of llm agents. In *ACL*. 8580–8622.