THE EVALUATION SUBSTRATE MATTERS: BENCHMARKING MULTI-AGENT SERVING SUBSTRATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent systems (MAS) usually coordinate heterogeneous models and tools to solve complex tasks, creating a growing need for reliable evaluation. However, a multi-agent workflow that is fastest in one serving environment can lose its advantage in another, even when its models, prompts, and task remain fixed. This makes a benchmark result conditional on how the workflow is executed. We introduce MASSYS, a benchmark for measuring this dependence in multi-agent large language model (LLM) systems. MASSYS executes the same candidate–task pairs across serving substrates and load regimes, using five workload families spanning software repair, research synthesis, financial analysis, interactive control, and diagram generation. It records task quality alongside delivery, latency, cost, resource pressure, and failure evidence. In a controlled motivation diagnostic, the Star workflow falls from first to fourth in mean-latency rank under KV-cache pressure, while Chain rises from fourth to first. The substrate sweeps further show that increasing load creates long runtime tails and timeout pressure. In several task families, quality among evaluated outputs changes less than effective quality over submitted jobs: the system delivers fewer usable results even when the surviving outputs remain competitive. Tool and runtime traces reveal that these outcomes involve both model serving and non-model execution. Together, the results motivate evaluating MAS designs over explicit operating conditions and reporting which tasks are delivered, how well they are solved, and where execution breaks down.

1 INTRODUCTION

Large language models (LLMs) are increasingly deployed as components inside agentic systems that plan, call tools, exchange intermediate results, and coordinate multiple specialized roles. Modern multi-agent systems (MAS) make this trend explicit: a user request may be decomposed by a planner, dispatched to code, search, finance, visualization, or critique agents, and then merged by a verifier or manager. Frameworks such as AutoGen and CAMEL popularized conversational and role-based multi-agent programming, while graph- and state-oriented frameworks make it practical to build long-running, tool-using, multi-actor workflows (Wu et al., 2024; Li et al., 2023). In parallel, agent benchmarks such as AgentBench, GAIA, SWE-Bench, and MultiAgentBench have shifted evaluation beyond static question answering toward interactive environments, tool use, software engineering, and collaboration or competition among agents (Liu et al., 2024; Mialon et al., 2024; Jimenez et al., 2024; Zhu et al., 2025).

This progress changes what a benchmark score means. For a multi-agent application, the score is no longer only a property of the underlying LLM: it also depends on topology, role prompts, model allocation, tool access, memory, error handling, and orchestration. Recent system-level evaluation work makes this point explicit. MASEval (Emde et al., 2026), for example, argues that the complete agentic system should be the unit of analysis, and shows that framework and implementation choices can materially affect multi-agent benchmark results. This is an important step beyond model-centric leaderboards. However, it leaves a lower layer largely implicit: the *serving substrate* on which the multi-agent system is executed. An agent can fail a coding benchmark before writing any solution code. On the Terminal-Bench task `bn-fit-modify`, Anthropic reports that installing a data-science stack can exhaust a tightly constrained container's memory. Across resource configurations, its Terminal-Bench 2.0 score differed by six percentage points with the same model, harness, and

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

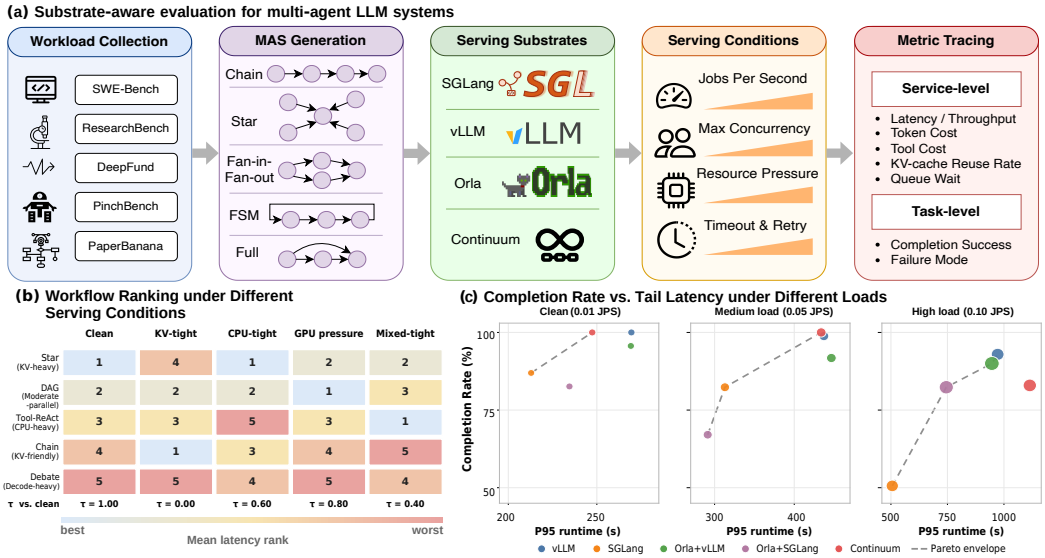


Figure 1: **Substrate-aware evaluation for multi-agent LLM systems.** (a) MASSYS fixes workloads and MAS candidates, then evaluates the same candidate-task pairs under different serving substrates and conditions. (b) A motivation diagnostic shows that workflow latency rankings change under controlled serving bottlenecks; Kendall's τ is reported only relative to the clean condition. (c) Real 4*H200 substrate-sweep samples show observed completion rate versus P95 runtime under increasing load, with point size indicating KV-cache pressure.

tasks (Segato, 2026). The execution environment can therefore determine whether a chosen solution strategy can even proceed.

The same issue appears across successive turns. Manus reports an average input-to-output token ratio of approximately 100:1, making reuse of accumulated context consequential for latency and cost (Ji, 2025). Continuum studies the related serving problem: an agent pauses to run a tool, and retaining its KV cache competes with admitting other requests (Li et al., 2025b). These observations motivate treating resource allocation, queueing, cache management, and tool execution as part of the conditions under which an agent is evaluated.

We use *serving substrate* to refer to the runtime, backend, request load, queueing policy, resource budget, cache behavior, timeout/retry policy, and orchestration mechanism that mediate execution between a MAS design and the underlying models and tools. Existing agent and MAS benchmarks typically fix this layer and attribute score differences to the model, scaffold, or candidate system. Fig. 1 illustrates why this is unsafe: the same workflow candidates change latency rankings under KV-cache, CPU, GPU, and mixed pressure, and serving cells move along different completion-latency frontiers as load increases.

We introduce MASSYS, a substrate-aware benchmark for multi-agent LLM systems. MASSYS draws tasks from SWE-Bench, ResearchBench, DeepFund, PinchBench [1], and PaperBanana, instantiates heterogeneous MAS candidates from a shared design space, and replays the same candidate-task pairs across interchangeable serving substrates and load conditions Jimenez et al. (2024); Yu et al. (2025); Li et al. (2025a); Zhu et al. (2026). This paired design holds workload and MAS design fixed while making backend, runtime, load, resource pressure, and timeout behavior explicit evaluation variables. Table 1 displays the position of MASSYS among existing work.

Our contributions are: (1) a substrate-aware benchmark protocol for MAS evaluation; (2) MASSYS, a mixed-domain benchmark with controlled candidates, interchangeable serving cells, and unified task/service traces; and (3) substrate-sweep evidence showing that backend/runtime choice, load, and workload composition change workflow rankings, tail latency, timeout behavior, tool pressure, and the observed reliability envelope.

[1] https://github.com/pinchbench/skill

Table 1: **Positioning of MASSys.** Comparison with representative benchmarks, frameworks, harnesses, and serving systems.

Line of work	Domain	Workload	Substr.	Stress	Dual	Trace
General evaluation suites: HELM, OpenCompass, Inspect-AI, lm-eval, LightEval (Liang et al., 2023; OpenCompass Contributors, 2023; UK AI Safety Institute, 2024; Biderman et al., 2024; Hugging Face, 2024)	✓	×	×	×	△	△
Agent and tool-use benchmarks: AgentBench, GAIA, SWE-Bench, WebArena, OSWorld, AppWorld, ToolBench, BFCL, τ^2-Bench (Liu et al., 2024; Mialon et al., 2024; Jimenez et al., 2024; Zhou et al., 2024; Xie et al., 2024; Trivedi et al., 2024; Qin et al., 2024; Patil et al., 2025; Barres et al., 2025)	✓	△	×	×	△	△
Multi-agent benchmarks: MultiAgentBench, MARBLE, ChatEval, ConVerse (Zhu et al., 2025; Chan et al., 2024; Gomaa et al., 2026)	✓	✓	×	×	△	△
Multi-agent frameworks: AutoGen, CAMEL, LangGraph, CrewAI, LlamaIndex, AgentVerse, AgentScope, MetaGPT, ChatDev (Wu et al., 2024; Li et al., 2023; LangChain, 2024a; CrewAI, 2024; Liu, 2022; Chen et al., 2024; Gao et al., 2024; Hong et al., 2024; Qian et al., 2024)	△	✓	×	×	×	×
Agent harnesses: AutoGenBench, HAL, AnyAgent, AgentGym, MCPEval (Fourney et al., 2024; Kapoor et al., 2026; Mozilla AI, 2025; Xi et al., 2024; LastMile AI, 2025)	△	△	△	△	△	△
LLM-app evaluation: DeepEval, Phoenix, TruLens, MLflow GenAI, LangSmith, RAGAs, Promptfoo, Langfuse, Weave (Ip and Vongthongsri, 2026; Arize AI, 2022; TruEra, 2020; Zaharia et al., 2018; LangChain, 2024b; Es et al., 2024; Promptfoo, 2024; Langfuse, 2024; Weights & Biases, 2024)	△	×	×	×	△	△
System-level MAS evaluation: MASEval, MAFBench, AgentTaxo (Emde et al., 2026; Orogat et al., 2026; Wang et al., 2025)	✓	✓	△	△	△	△
Generic LLM serving: vLLM, SGLang, Orca, DistServe, Sarathi-Serve, TensorRT-LLM, TGI, LightLLM (Kwon et al., 2023; Zheng et al., 2024; Yu et al., 2022; Zhong et al., 2024; Agrawal et al., 2024; NVIDIA, 2023; Hugging Face, 2023; ModelTC Team, 2023)	×	×	✓	✓	△	△
Agentic/application serving: Parrot, Ayo, Orla, Continuum, Helium, Nalar, Autellix, SDAS (Lin et al., 2024; Tan et al., 2025; Shahout et al., 2026; Li et al., 2025b; Wadlom et al., 2026; Laju et al., 2026; Luo et al., 2026; Agarwal et al., 2026)	△	△	✓	✓	△	△
MASSys	✓	✓	✓	✓	✓	✓

Legend. ✓ = primary objective; △ = partially covered or indirectly available; × = not a primary evaluation dimension. *Dimensions.* Domain = multi-domain coverage; Workload = variation of topology, agent count, prompts, models, and tools; Substr. = backend/runtime as explicit variable; Stress = load, queueing, timeout, failure, or resource pressure; Dual = task- and service-level metrics; Trace = failure-mode tracing under paired changes.

2 Related Work

Agent and MAS benchmarks. Agent benchmarks have moved LLM evaluation from static question answering to executable tasks with tools, environments, and long-horizon interaction. AgentBench and GAIA evaluate general agent abilities; SWE-Bench, WebArena, OSWorld, AppWorld, ToolBench, BFCL, and τ^2-Bench stress software repair, web or desktop control, app-level actions, tool/API use, function calling, and conversational control (Liu et al., 2024; Mialon et al., 2024; Jimenez et al., 2024; Zhou et al., 2024; Xie et al., 2024; Trivedi et al., 2024; Qin et al., 2024; Patil et al., 2025; Barres et al., 2025). Multi-agent benchmarks add collaboration, competition, debate, safety, and organization structure through datasets such as MultiAgentBench, ChatEval, ConVerse, and AgentTaxo (Zhu et al., 2025; Chan et al., 2024; Gomaa et al., 2026; Wang et al., 2025). These works provide the task side of our benchmark, but they usually hold the execution substrate fixed. MASSys keeps the MAS candidate and task distribution fixed while varying the backend, runtime, load, queueing, and resource conditions that execute the candidate.

Evaluation infrastructure and system-level MAS evaluation. General evaluation suites such as HELM, OpenCompass, Inspect-AI, lm-eval, and LightEval improve reproducibility and comparability for language-model evaluation (Liang et al., 2023; OpenCompass Contributors, 2023; UK AI Safety Institute, 2024; Biderman et al., 2024; Hugging Face, 2024). LLM-application tools such as DeepEval, Phoenix, TruLens, MLflow GenAI, LangSmith, RAGAs, Promptfoo, Langfuse, and Weave add tracing, regression tests, observability, and task-specific metrics (Ip and Vongthongsri, 2026; Arize AI, 2022; TruEra, 2020; Zaharia et al., 2018; LangChain, 2024b; Es et al., 2024; Promptfoo, 2024; Langfuse, 2024; Weights & Biases, 2024). Agent harnesses including AutoGenBench, HAL, AnyAgent, AgentGym, and MCPEval reduce the cost of repeated runs, sandboxing, framework integration, and tool or MCP evaluation (Fourney et al., 2024; Kapoor et al., 2026; Mozilla AI, 2025; Xi et al., 2024; LastMile AI, 2025). The closest work is MASEval, which treats the complete multi-agent system—framework, topology, orchestration, tools, and model choices—as the evaluation unit; MAFBench similarly studies framework-level trade-offs (Emde et al., 2026; Orogat et al., 2026).

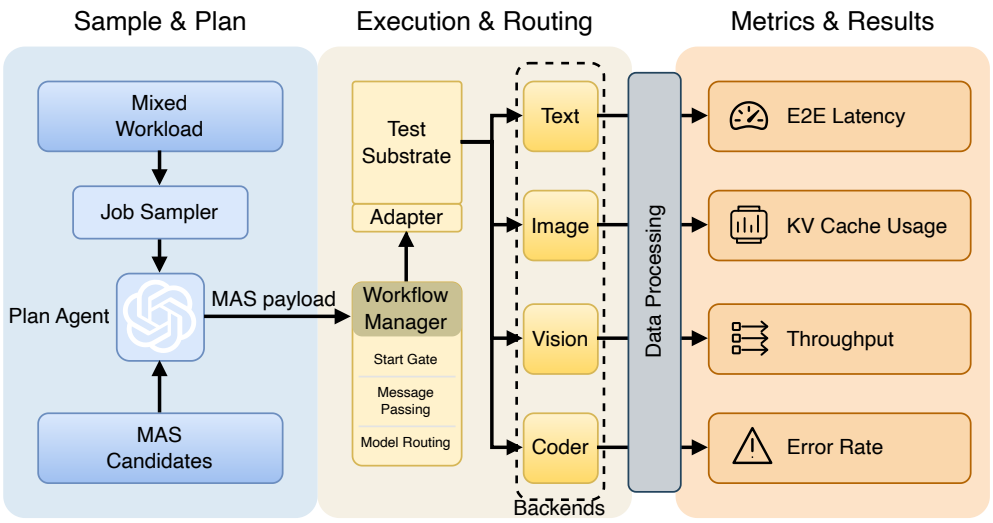


Figure 2: **MASSys evaluation architecture.** MASSys samples mixed workloads and MAS candidates, converts them into a structured MAS payload through a plan agent, and executes the payload inside a substrate-aware routing layer. The workflow manager coordinates start gates, message passing, and model routing across text, image, vision, and coding backends. The resulting traces are processed into service-level metrics such as end-to-end latency, KV-cache usage, throughput, and error rate, enabling controlled comparison of MAS behavior across serving substrates.

MASSys adopts this system-level view and moves one layer lower: after the MAS candidate is fixed, it measures whether substrate choices change task outcomes, service behavior, and failure evidence.

LLM serving and agentic/application serving. LLM serving systems make backend behavior a primary systems problem. vLLM, SGLang, Orca, DistServe, Sarathi-Serve, TensorRT-LLM, TGI, and LightLLM optimize memory management, structured-program execution, batching, prefill/decode scheduling, throughput, and latency (Kwon et al., 2023; Zheng et al., 2024; Yu et al., 2022; Zhong et al., 2024; Agrawal et al., 2024; NVIDIA, 2023; Hugging Face, 2023; ModelTC Team, 2023). Recent application and agentic-serving work, including Parrot, Ayo, Teola, Orla, Continuum, Helium, Nalar, Agentix/Autellix, and software-defined agentic serving, further targets workflow structure, KV-cache reuse, scheduling, and runtime control for multi-call or multi-turn applications (Lin et al., 2024; Tan et al., 2025; 2024; Shahout et al., 2026; Li et al., 2025b; Wadlom et al., 2026; Laju et al., 2026; Luo et al., 2026; Agarwal et al., 2026). These systems optimize how workflows are served. MASSys asks the evaluation question that follows: when a MAS benchmark is run on such substrates, which measured effects come from the candidate, the workload, or the substrate itself?

3 MASSys: Substrate-aware MAS Evaluation Benchmark

MASSys is a benchmark and evaluation framework for studying how serving substrates affect multi-agent LLM systems. The benchmark is organized around three principles. First, the evaluated object is a complete MAS candidate, including its topology, number of agents, role prompts, model allocation, tools, and orchestration policy. Second, a serving substrate is a first-class experimental variable, covering backend/runtime choice, request load, queueing behavior, resource pressure, cache behavior, and scheduling policy. Third, every run records both task-level and service-level outcomes, so that a completed execution is not mistaken for a correct solution and a low-latency execution is not mistaken for a robust MAS design. Fig. 2 summarizes the evaluation pipeline that connects workload sampling, MAS candidate construction, substrate execution, and trace processing.

MASSys evaluates complete MAS candidates under controlled serving substrates. Each candidate includes topology, agent count, role prompts, model assignment, tool access, memory, and orchestration policy. Each substrate cell specifies backend/runtime, offered load, queueing behavior, resource

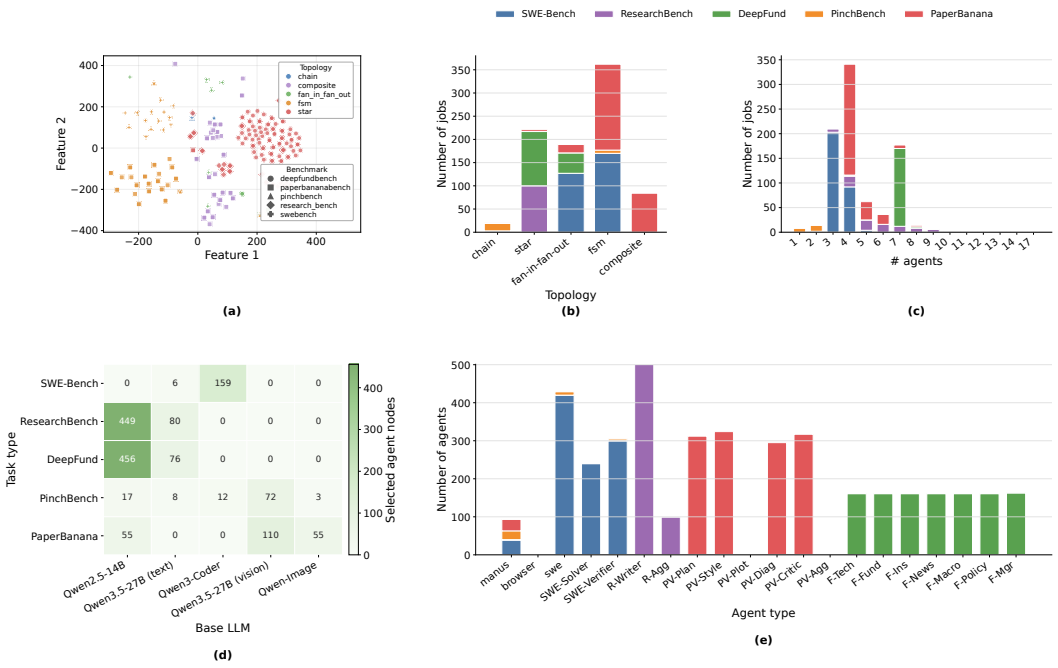


Figure 3: **MAS candidate topology, agent-role and base-LLM distribution.**

budget, cache policy, and timeout/retry policy. Every run records both task-level and service-level outcomes so that a completed execution is not mistaken for a correct answer and a low-latency execution is not mistaken for a robust MAS design.

3.1 Workloads and Candidates

MASSys uses five task families: SWE-Bench, ResearchBench, DeepFund, PinchBench, and PaperBanana. We treat them as a mixed service workload rather than independent leaderboards. Each family has sampling probability 0.2; after selecting a family, the runner samples one task uniformly from its pool with replacement. The mixture covers software repair, research synthesis, financial analysis, interactive control, and diagram-oriented generation, producing jobs with different tool chains, fan-out/fan-in structures, verifier loops, and modality-specific routing. The candidate space covers chain, star, fan-in/fan-out, finite-state-machine, and composite workflows. Fig. 4 summarizes the candidate task pool, while Fig. 3 summarizes the resulting MAS structure, aggregate agent-role distribution and generated topology diversity. Fig. 3 further shows that model allocation is task-dependent rather than drawn from a single homogeneous base model. Together, they show that MASSys varies both benchmark source and candidate topology rather than evaluating a single prompt or role template.

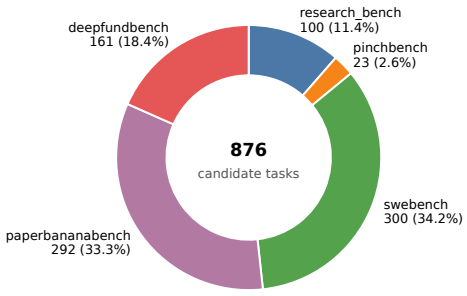


Figure 4: **Candidate task pool composition.** The pool contains 876 candidate tasks across five workload families.

Table 2: **MASSYS Metrics.**

Service/System Metric	Definition
Completion	Fraction of submitted jobs that finish before the 30-minute submission window closes.
Execution success	Fraction of submitted jobs whose worker returns a usable final artifact, independent of task correctness.
Timeout/killed/failure	Fractions normalized by submitted jobs; killed jobs include deadline-terminated runs after the drain period.
Runtime and queueing	Mean/P95 end-to-end runtime and queue wait, computed over non-killed execution-success jobs.
Cost and pressure	Token usage, tool latency, concurrency, and KV-cache pressure; KV-cache pressure is reported from backend samples as mean and P95.
Task-Quality Metric	
SWE-Bench	Docker harness resolved rate: resolved instances divided by evaluated instances.
ResearchBench	Embedding cosine similarity between generated and reference research artifacts, averaged over evaluated jobs.
DeepFund	Trading outcome score derived from realized or next-trading-day PnL labels: profit, flat, or loss.
PinchBench	Combined score from automated graders and LLM-judge rubric scores, using the task's grading weights.
PaperBanana	Overall visual score, computed as the mean of instruction faithfulness, conciseness, readability, and aesthetics judge scores, divided by 10.

3.2 PAIRED SUBSTRATE SWEEPS

MASSYS executes the same candidate–task pairs across vLLM-style serving, SGLang-style serving, Continuum-style serving, Orla over vLLM, and Orla over SGLang. The sweeps vary offered load and hardware regime while keeping the workload distribution fixed. This paired design makes the substrate an experimental variable: a change in completion, latency, timeout behavior, or failure evidence can be attributed to the serving condition rather than to an accidental change in task sampling.

3.3 TRACING AND METRICS

The trace layer records job status, end-to-end runtime, unfinished-at-window-close evidence, graph structure, tool calls and tool latencies, LLM-call counts, queue wait, token usage, concurrency, resource samples, and KV-cache behavior. Post-hoc evaluators compute task quality using benchmark-specific checks, including execution-based grading, structured validation, and judge-based scoring when needed. Table 2 summarizes the metric axes. This separation allows MASSYS to distinguish task quality, service feasibility, and failure evidence within the same run.

4 EXPERIMENTS

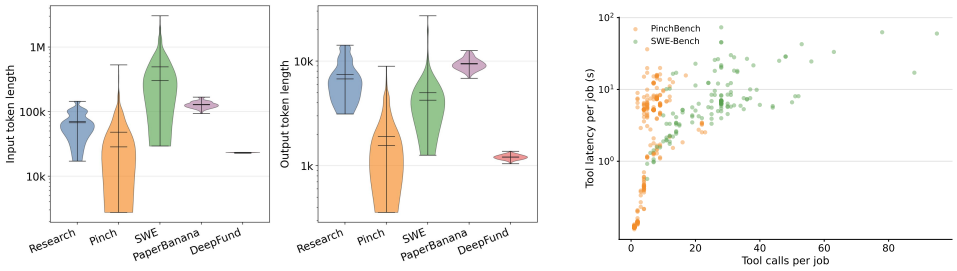
We evaluate MASSYS on mixed MAS workloads under controlled substrate sweeps. The goal of this section is not to produce a single leaderboard of MAS candidates, but to characterize how backend, runtime, load, and orchestration conditions change the feasibility and trace behavior of the same class of multi-agent jobs. Unless otherwise stated, *succeeded* denotes execution success: the job completed and returned a usable final artifact under the service condition.

4.1 EVALUATION SETTINGS

We evaluate MASSYS with paired substrate sweeps over five serving cells: vLLM-style serving, SGLang-style serving, Continuum-style serving, Orla over vLLM, and Orla over SGLang. Table 3 summarizes the hardware-specific model allocation; across both settings, MAS plans are generated by a GPT-5.4 plan agent, and jobs are submitted by a Poisson arrival process at offered loads of

Table 3: **Hardware and model sets for substrate sweeps.**

Hardware	Language Models	Vision Models	Image Models
2×H200	Qwen3.5-27B	Qwen3.5-27B	Qwen-image-2512
4×H200	Qwen2.5-14B-Instruct; Qwen3.5-27B; Qwen3-Coder-30B-A3B-Instruct	Qwen3.5-27B	Qwen-image-2512



(a) Token length. (b) Tool latency vs. tool calls.

Figure 5: **Workload heterogeneity in token and tool demand.** (a) Input and output token length distributions vary substantially across task families, indicating different prompt/context sizes and generation scales. (b) Per-job tool latency generally increases with the number of tool calls but exhibits heavy tails, especially for SWE-Bench jobs with many edit, execution, and verification steps. These workload-level differences explain why mixed MAS evaluation stresses serving substrates differently from single-domain or single-request benchmarks.

0.005, 0.01, 0.02, 0.05, and 0.10 JPS over a 30-minute window with seed 42 and a 1200 s deadline. Unless otherwise stated, *success* denotes execution success—the job completed and returned a usable final artifact—not task accuracy. We therefore report completion, success, mean and P95 runtime, KV-cache pressure, timeout behavior, run-completion elapsed time, and concurrency as separate service metrics.

4.2 Main Results

Result 1: Load turns MAS serving into a tail-latency problem. Increasing load affects the entire MAS execution graph, not only isolated model calls. Fig. 6 shows that, as offered load increases, completion and execution-success rates decline in several substrate cells, while mean job runtime, P95 runtime, timeout rate, run-completion elapsed time, KV-cache pressure, and P95 concurrency increase. The effect is visible in both 2×H200 and 4×H200 regimes. This pattern matters for MAS evaluation because multi-agent jobs contain multiple dependent calls, tool invocations, and aggregation steps. A delayed branch can block downstream agents or the final response, so tail behavior is part of the benchmark outcome rather than a secondary serving statistic.

Result 2: backend/runtime choices change the service envelope. Fig. 1(c) and Fig. 6 show that substrate choice changes the completion-latency-reliability trade-off observed by the same mixed MAS workload. Fig. 7 shows that this service envelope also changes reported task quality. Effective quality is most sensitive in task families where serving failures remove many jobs from the evaluated set, with high-load drops especially visible for SWE-Bench, PinchBench, and PaperBanana in several 2×H200 cells. By contrast, evaluated-only quality is often flatter for ResearchBench and parts of DeepFund, indicating that many completed-and-evaluated artifacts retain similar quality even as fewer jobs survive the service condition. The 4×H200 regime mitigates some effective-quality loss but does not remove substrate dependence, especially for visual and interactive workloads. Some serving cells achieve lower runtime over portions of the load range, while others maintain stronger completion or execution-success behavior under the same offered load. The comparison is therefore not a single total ordering of backends. Instead, MASSYS exposes a service envelope: a substrate may be attractive for latency among completed jobs but less favorable in reliability, timeout behavior, or backlog under heterogeneous mixed workloads. A deployment-relevant MAS benchmark should report these axes separately.

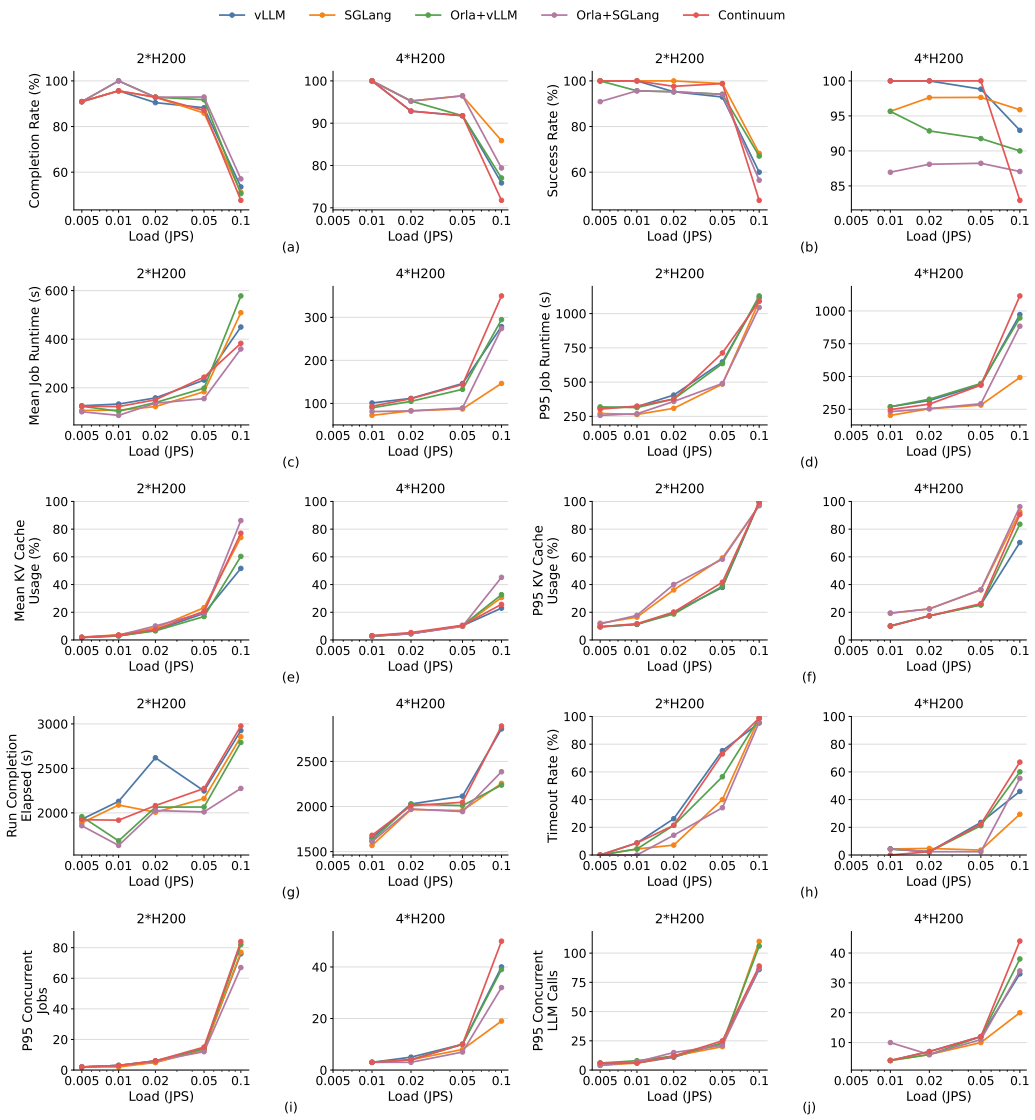


Figure 6: **Load-dependent service behavior across serving substrates.** Each pair of panels plots a service metric versus offered load for the 2×H200 and 4×H200 regimes, with color indicating the serving substrate. Completion and execution-success rates remain high at light load but degrade under stress, while mean and P95 runtime, KV-cache pressure, timeout rate, run-completion elapsed time, and P95 concurrency rise sharply as load increases. The figure shows that substrate comparisons are multi-dimensional: a serving cell may be fast at moderate load yet accumulate backlog, KV pressure, or failures under heavy mixed-MAS traffic.

Result 3: workload composition changes the measured serving profile. Workload composition is itself a substrate-sensitive variable. Fig. 5 shows that task families differ in both input and output token length distributions, and that tool latency increases with the number of tool calls while exhibiting heavy tails. Fig. 8 further decomposes the tool layer. Read/view, text-search, Python, test, and edit operations dominate call volume, but tool categories differ substantially in duration and success behavior. Browser and execution/testing paths are especially important because they combine non-trivial latency with more variable reliability. These observations explain why a mixed MAS workload can produce service behavior that is not predicted by agent count or request rate alone. Token distribution, tool mix, topology, and role allocation must be reported together with latency and completion metrics.

Result 4: trace decomposition identifies where substrate effects arise. Trace metrics reveal that the same final runtime can be caused by different mechanisms. Fig. 9 decomposes mean job runtime for the 2×H200 vLLM and SGLang cells into planning, backend prefill, backend decode, backend queue wait, backend other time, tool execution, and residual orchestration overhead. At light and moderate load, runtime is dominated by backend decode together with orchestration and tool overhead. At the highest load, backend queue wait and residual overhead become prominent. Combined with the tool-layer evidence in Fig. 8, this decomposition shows why MASSYS records traces rather than only scalar scores: slow or failed MAS executions can arise from backend decoding, queueing, tool calls, orchestration overhead, or their interaction along the MAS graph.

4.3 DISCUSSION

Fast substrates and robust substrates are not always the same. The results separate properties that are often conflated in benchmark reporting. A serving cell can have favorable runtime among completed jobs while showing weaker completion, success, timeout, or backlog behavior under heavier load. Conversely, a substrate with higher latency may preserve a more stable service outcome in part of the operating range. This motivates reporting MAS benchmark results as service envelopes rather than as a single backend or candidate score.

Backlog is an evaluation outcome. The fixed service-window view makes queue growth and unfinished work visible. Under heavy load, Fig. 6 shows that timeout rate, run-completion elapsed time, and concurrent-job pressure increase along with tail runtime. For interactive MAS applications, this is not a bookkeeping artifact. A delayed worker branch can block a verifier, a delayed verifier can block the final aggregation step, and a delayed final answer can turn an otherwise successful execution into an unusable service result. MASSYS therefore treats deadline behavior and unfinished-at-window-close evidence as first-class service metrics.

The tool layer is part of MAS serving. Fig. 8 shows that tool calls are neither uniform nor negligible. Some tools are frequent and lightweight; others are less frequent but dominate latency tails or reliability variation. This matters because MAS candidates differ in how they allocate work across search, reading, editing, execution, browser control, and verification. A model-only benchmark would miss this source of variation and may incorrectly attribute tool-layer effects to model quality or agent topology.

Failure attribution should be part of the benchmark report. Table **??** separates task outcome, service outcome, and failure evidence. This separation is necessary for substrate-aware evaluation. A failed MAS run may reflect task reasoning, agent coordination, tool behavior, environment constraints, queueing/timeout, resource pressure, backend/runtime behavior, or judge/format issues. MASSYS therefore records trace evidence rather than collapsing all unsuccessful executions into a generic failure label. This design makes it possible to distinguish service-induced failures from model, tool, environment, or task-specific failures.

5 CONCLUSION

We presented MASSYS, a substrate-aware benchmark for multi-agent LLM systems. MASSYS executes controlled MAS candidates over mixed-domain workloads under paired serving substrates and records task outcomes, service outcomes, and trace-derived failure evidence. The reported sweeps show that workflow rankings change under resource bottlenecks, load creates heavy-tail runtime and timeout pressure, and serving cells expose different completion–latency–reliability envelopes. The workload and trace analyses further show that token distribution, tool mix, queueing, and orchestration overhead are essential parts of the benchmark outcome. These findings suggest that future MAS benchmarks should report substrate, load, hardware, workload structure, and trace evidence together with model and topology choices.

ETHICS STATEMENT

This work collects no new human-subject data or private user interactions; its workloads derive from public benchmarks and data sources. Released traces are screened for credentials and deployment-specific identifiers, but may contain benchmark inputs and model-generated artifacts subject to upstream terms. DeepFund outputs are not financial advice, model-judge scores may inherit evaluator bias, and code/tool execution should use isolated environments with scoped credentials.

AI USE STATEMENT

In this work, we used generative AI tools to polish writing, retrieval and discovery, such as finding related work. We have not used generative AI tools for generating synthetic datasets, and proving mathematical claims are not applicable to this work. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Sections 3 and 4 describe the benchmark protocol, workload construction, and experimental settings. Appendix A documents the execution, model and decoding, software, and evaluator configurations, together with known limitations to exact replication. The accompanying code release provide the implementation, configurations, and execution artifacts used to derive the reported results.

REFERENCES

Saurabh Agarwal, Marco Laju, Jayanth Srinivasa, Myungjin Lee, and Aditya Akella. Software-defined agentic serving. *arXiv preprint arXiv:2601.03197*, 2026. doi: 10.48550/arXiv.2601.03197. URL https://arxiv.org/abs/2601.03197.

Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, and Ramachandran Ramjee. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/agrawal.

Arize AI. Phoenix: AI observability and evaluation. https://github.com/Arize-ai/phoenix, 2022. URL https://github.com/Arize-ai/phoenix.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. Lessons from the trenches on reproducible evaluation of language models. *arXiv preprint arXiv:2405.14782*, 2024. URL https://arxiv.org/abs/2405.14782.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. ChatEval: Towards better LLM-based evaluators through multi-agent debate. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=FQepisCUWu.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=EHg5GDnyq1.

CrewAI. CrewAI. https://github.com/crewAIInc/crewAI, 2024. URL https://github.com/crewAIInc/crewAI.

Cornelius Emde, Alexander Rubinstein, Anmol Goel, Ahmed Heakl, Sangdoo Yun, Seong Joon Oh, and Martin Gubri. MASEval: Extending multi-agent evaluation from models to systems, 2026. URL https://arxiv.org/abs/2603.08835.

Shahul Es, Jithin James, Luis Espinosa-Anke, and Steven Schockaert. RAGAs: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 150–158. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.eacl-demo.16/.

Adam Fourney, Qingyun Wu, Gagan Bansal, and Chi Wang. AutoGenBench: A tool for measuring and evaluating autogen agents. Microsoft AutoGen Blog, 2024. URL https://microsoft.github.io/autogen/0.2/blog/2024/01/25/AutoGenBench/.

Dawei Gao, Zitao Li, Xuchen Pan, Weirui Kuang, Zhijian Ma, Bingchen Qian, Fei Wei, Wenhao Zhang, Yuexiang Xie, Daoyuan Chen, Liuyi Yao, Hongyi Peng, Zeyu Zhang, Lin Zhu, Chen Cheng, Hongzhu Shi, Yaliang Li, Bolin Ding, and Jingren Zhou. AgentScope: A flexible yet robust multi-agent platform. *arXiv preprint arXiv:2402.14034*, 2024. URL https://arxiv.org/abs/2402.14034.

Amr Gomaa, Ahmed Salem, and Sahar Abdelnabi. ConVerse: Benchmarking contextual safety in agent-to-agent conversations. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 3246–3268, Rabat, Morocco, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-eacl.170. URL https://aclanthology.org/2026.findings-eacl.170/.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Zhang, Ceyao Wang, Zili Wang, Steven K. Yau, Zijuan Lin, Liyang Zhou, Chenkai Ran, Lingfeng Xiao, Chenglin Wu, and J"urgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

Hugging Face. Text generation inference. https://github.com/huggingface/text-generation-inference, 2023. Software package.

Hugging Face. LightEval: A lightweight evaluation suite for large language models. https://github.com/huggingface/lighteval, 2024. Software package.

Jeffrey Ip and Kritin Vongthongsri. DeepEval: The open-source LLM evaluation framework. https://github.com/confident-ai/deepeval, 2026. URL https://github.com/confident-ai/deepeval.

Yichao Ji. Context engineering for AI agents: Lessons from building Manus. Manus, July 2025. URL https://manus.im/blog/Context-Engineering-for-AI-Agents-Lessons-from-Building-Manus. Published July 18, 2025.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary S. Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, Franck Ndzomga, Dheeraj Oruganty, Sophie Luskin, Kangheng Liu, Botao Yu, Amit Arora, Dongyoon Hahm, Harsh Trivedi, Huan Sun, Juyong Lee, Tengjun Jin, Yifan Mai, Yifei Zhou, Yuxuan Zhu, Rishi Bommasani, Daniel Kang, Dawn Song, Peter Henderson, Yu Su, Percy Liang, and Arvind Narayanan. Holistic agent leaderboard: The missing infrastructure for AI agent evaluation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=vUaY1t64ZZ.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles*, pages 611–626, 2023. doi: 10.1145/3600006.3613165. URL https://arxiv.org/abs/2309.06180.

Marco Laju, Donghyun Son, Saurabh Agarwal, Nitin Kedia, Myungjin Lee, Jayanth Srinivasa, and Aditya Akella. Nalar: An agent serving framework. *arXiv preprint arXiv:2601.05109*, 2026. doi: 10.48550/arXiv.2601.05109. URL https://arxiv.org/abs/2601.05109.

LangChain. LangGraph: Low-level orchestration framework for stateful agents. https://github.com/langchain-ai/langgraph, 2024a. URL https://github.com/langchain-ai/langgraph.

LangChain. LangSmith: LLM application evaluation and observability platform. https://docs.smith.langchain.com/, 2024b. URL https://docs.smith.langchain.com/.

Langfuse. Langfuse: Open-source llm engineering platform. https://github.com/langfuse/langfuse, 2024. Software package.

LastMile AI. MCP-Eval: Lightweight eval framework for MCP servers. https://github.com/lastmile-ai/mcp-eval, 2025. URL https://github.com/lastmile-ai/mcp-eval.

Changlun Li, Yao Shi, Chen Wang, Qiqi Duan, Runke Ruan, Weijie Huang, Haonan Long, Lijun Huang, Nan Tang, and Yuyu Luo. Time travel is cheating: Going live with deepfund for real-time fund investment benchmarking, 2025a. URL https://arxiv.org/abs/2505.11065.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=3IyL2XWDkG.

Hanchen Li, Runyuan He, Qiuyang Mang, Qizheng Zhang, Huanzhi Mao, Xiaokun Chen, Hangrui Zhou, Huanchen Zhang, Alvin Cheung, Joseph Gonzalez, and Ion Stoica. Continuum: Efficient and robust multi-turn LLM agent scheduling with KV cache time-to-live, 2025b. URL https://arxiv.org/abs/2511.02230v7. First submitted November 4, 2025; version 7, September 8, 2026.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. Holistic evaluation of language models. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=iO4LZibEqW.

Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. Parrot: Efficient serving of LLM-based applications with semantic variable. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 929–945, Santa Clara, CA, 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/lin-chaofan.

Jerry Liu. LlamaIndex. https://github.com/run-llama/llama_index, 2022. URL https://github.com/run-llama/llama_index.

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. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=zAdUB0aCTQ.

Michael Luo, Xiaoxiang Shi, Colin Cai, Tianjun Zhang, Justin Wong, Yichuan Wang, Chi Wang, Yanping Huang, Zhifeng Chen, Joseph E. Gonzalez, and Ion Stoica. Agentix: An efficient serving engine for LLM agents as general programs. In *23rd USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 2026. URL https://www.usenix.org/conference/nsdi26/presentation/luo.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=fibxvahvs3.

ModelTC Team. LightLLM: A lightweight and high-performance inference and serving framework. https://github.com/ModelTC/lightllm, 2023. Software package.

Mozilla AI. any-agent: A single interface to use and evaluate agents across frameworks. https://github.com/mozilla-ai/any-agent, 2025. URL https://github.com/mozilla-ai/any-agent.

NVIDIA. TensorRT-LLM. https://github.com/NVIDIA/TensorRT-LLM, 2023. Software package.

OpenCompass Contributors. OpenCompass: A universal evaluation platform for foundation models. https://github.com/open-compass/opencompass, 2023. URL https://github.com/open-compass/opencompass.

Abdelghny Orogat, Ana Rostam, and Essam Mansour. Understanding multi-agent LLM frameworks: A unified benchmark and experimental analysis, 2026. URL https://arxiv.org/abs/2602.03128.

Shishir G. Patil, Tianyi Zhang, Xin Wang, and Joseph E. Gonzalez. The berkeley function calling leaderboard. In *Proceedings of Machine Learning Research*, 2025. URL https://gorilla.cs.berkeley.edu/leaderboard.html.

Promptfoo. Promptfoo: Open-source evaluation and red teaming for llm applications. https://github.com/promptfoo/promptfoo, 2024. Software package.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 15174–15186. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.acl-long.810/.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Gian Segato. Quantifying infrastructure noise in agentic coding evals. Anthropic Engineering, February 2026. URL https://www.anthropic.com/engineering/infrastructure-noise. Published February 5, 2026.

Rana Shahout, Hayder Tirmazi, Minlan Yu, and Michael Mitzenmacher. Orla: A library for serving llm-based multi-agent systems. *arXiv preprint arXiv:2603.13605*, 2026. doi: 10.48550/arXiv.2603.13605. URL https://arxiv.org/abs/2603.13605.

Xin Tan, Yimin Jiang, Yitao Yang, and Hong Xu. Teola: Towards end-to-end optimization of LLM-based applications. *arXiv preprint arXiv:2407.00326*, 2024. doi: 10.48550/arXiv.2407.00326. URL https://arxiv.org/abs/2407.00326.

Xin Tan, Yimin Jiang, Yitao Yang, and Hong Xu. Towards end-to-end optimization of LLM-based applications with Ayo. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '25, pages 1302–1316. Association for Computing Machinery, 2025. doi: 10.1145/3676641.3716278. URL https://doi.org/10.1145/3676641.3716278.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 16022–16076. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.acl-long.850/.

TruEra. TruLens: Evaluation and tracking for LLM experiments and AI agents. https://github.com/truera/trulens, 2020. URL https://github.com/truera/trulens.

UK AI Safety Institute. Inspect AI: Framework for large language model evaluations. https://github.com/UKGovernmentBEIS/inspect_ai, 2024. URL https://github.com/UKGovernmentBEIS/inspect_ai.

Noppanat Wadlom, Junyi Shen, and Yao Lu. Efficient LLM serving for agentic workflows: A data systems perspective. *arXiv preprint arXiv:2603.16104*, 2026. doi: 10.48550/arXiv.2603.16104. URL https://arxiv.org/abs/2603.16104.

Qingyun Wang et al. AgentTaxo: Dissecting and benchmarking taxonomy of large language model-based multi-agent systems. *OpenReview preprint*, 2025. URL https://openreview.net/forum?id=0iLbiYYIpC.

Weights & Biases. Weave: A toolkit for evaluating and tracing ai applications. https://github.com/wandb/weave, 2024. Software package.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. AgentGym: Evolving large language model-based agents across diverse environments, 2024. URL https://arxiv.org/abs/2406.04151.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Jun Zhou, Dongchan Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2404.07972.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation*, pages 521–538. USENIX Association, 2022. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Haofei Yu, Zhaochen Hong, Zirui Cheng, Kunlun Zhu, Keyang Xuan, Jinwei Yao, Tao Feng, and Jiaxuan You. Researchtown: Simulator of human research community, 2025. URL https://arxiv.org/abs/2412.17767.

Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Fen Xie, and Corey Zumar. Accelerating the machine learning lifecycle with MLflow. *IEEE Data Engineering Bulletin*, 41(4):39–45, 2018. URL https://api.semanticscholar.org/CorpusID:83459546.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=VqkAKQibpq.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/zhong.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=oKn9c6ytLx.

Dawei Zhu, Rui Meng, Yale Song, Xiyu Wei, Sujian Li, Tomas Pfister, and Jinsung Yoon. Paperbanana: Automating academic illustration for ai scientists, 2026. URL https://arxiv.org/abs/2601.23265.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. MultiAgentBench: Evaluating the collaboration and competition of LLM agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8580–8622. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.421. URL https://aclanthology.org/2025.acl-long.421/.

A REPRODUCIBILITY

Our code is available at `https://anonymous.4open.science/r/MAS-Sys-2D7A`

Execution protocol. Each substrate cell is represented by one run ($n = 1$). The runner releases jobs at the frozen arrival offsets without a benchmark-level concurrency semaphore; native cells rely on the backend scheduler, while Orla uses per-backend concurrency and queue capacities of 1,000, chosen not to add a smaller front-end admission cap. The 30-minute window stops new arrivals and defines completion, but is not a global kill deadline: already submitted jobs drain until termination, subject to a 1,200 s per-job deadline. Model-call transport errors are retried for at most six attempts with randomized exponential backoff between 1 and 60 s.

Models and decoding. Each served model instance is deployed within one H200 (tensor parallelism one). We use BF16 weights without quantization and do not set an additional backend context-length cap, so the model/backend native context limit applies. Language and vision agents use temperature 0, a 4,096-token output cap, and disabled Qwen thinking mode; Qwen-image uses its image-generation defaults with temperature 1.0. The PlanAgent uses temperature 0 and an 8,192-token output cap. Backends start fresh for each run without an explicit warm-up, after which prefix caches accumulate within the run.

Software and artifacts. The released setup uses Python 3.11 and ORLA 1.2.15. The runtime environments use PyTorch/Transformers 2.11.0/5.6.2 for vLLM, 2.9.1/5.5.4 for SGLang, 2.10.0/4.57.6 for vLLM-Omni, and 2.8.0/5.6.2 for Continuum; the corresponding PyTorch builds target CUDA 13.0 or 12.8. vLLM is `0.19.2.dev0+gf150107.local` (commit `f150107`); the custom SGLang, vLLM-Omni, and Continuum source trees are included in the code release. Model identifiers are listed in Table 3.

Task-quality evaluators. ResearchBench uses `text-embedding-3-large` with 1,024-dimensional embeddings and cosine similarity. PinchBench uses the task's released automated grader, LLM rubric, and grading weights; its LLM judge is GPT-4o at temperature 0 and receives the task prompt, execution transcript, and task-specific rubric. PaperBanana uses GPT-4o at temperature 0 with the released PaperBanana prompts to score instruction faithfulness, conciseness, readability, and aesthetics on a 0–10 scale. SWE-Bench uses its per-instance Docker test harness. DeepFund maps the stored realized/next-trading-day outcome to profit/flat/loss and then to $1/0.5/0$. Evaluator prompts, rubrics, inputs, and structured outputs are retained in the code or trace releases.

Service aggregates are deterministically recomputable from the released traces. Repeating DeepFund against live data sources can change the market snapshot; the paper therefore uses the stored outcome labels. Exact commands, sanitized configurations, and artifact schemas are documented with the code and trace releases.

B LIMITATIONS

MASSYS is intended to expose substrate effects rather than to exhaustively rank all MAS frameworks or serving systems. The current benchmark covers five workload families and a controlled set of MAS topologies, but it does not represent the full space of deployed agent applications, especially long-running enterprise workflows, human-in-the-loop tasks, and safety-critical interactive settings. The reported substrate sweeps also focus on a finite set of serving cells and hardware regimes; additional backends, schedulers, accelerators, and production admission-control policies may expose different trade-offs. Moreover, MASSYS separates execution success from task quality, but task-quality evaluation remains benchmark-specific and can depend on the fidelity of execution-based checks, structured validators, or judge models. Finally, failure attribution is trace-based rather than causal: the traces identify evidence for queueing, tool, resource, backend, or judge/format failures, but they may not fully isolate root causes when multiple failures interact along the same MAS graph. These limitations motivate extending MASSYS with broader workload sources, additional serving substrates, repeated runs under randomized seeds, and stronger task-specific evaluators.

C Additional Task-Quality and Trace Results

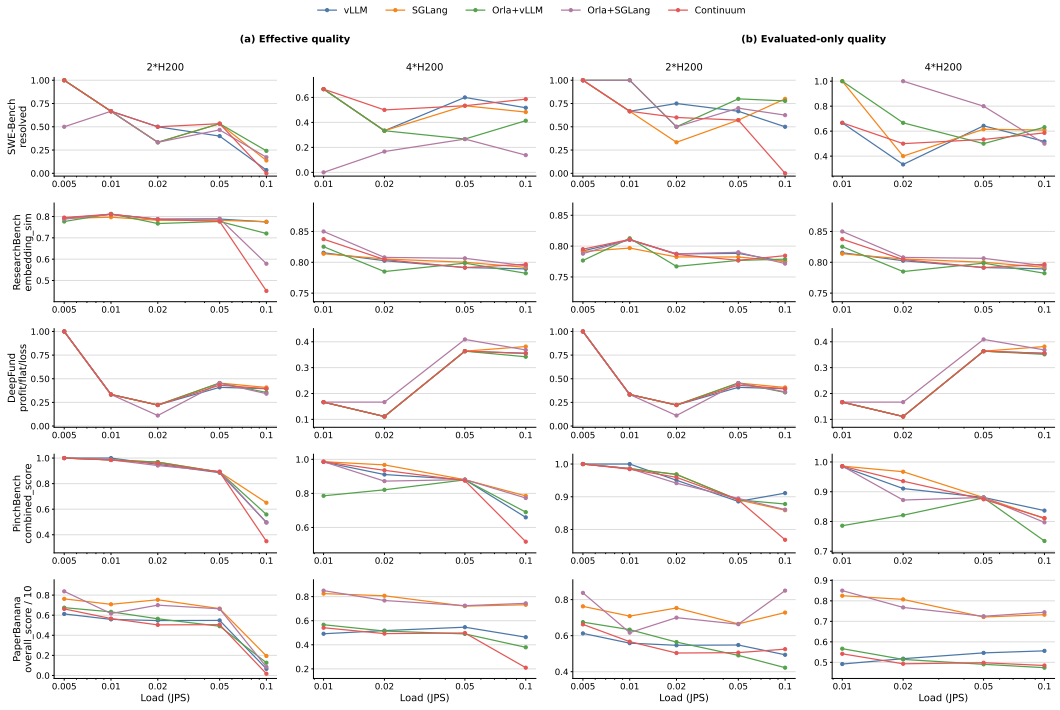


Figure 7: **Task quality across load and serving configurations.** The left half reports effective quality, where failed, timed-out, or unevaluated jobs contribute zero, and the right half reports evaluated-only quality over jobs with successful post-hoc evaluation. Rows use each benchmark's normalized quality metric, columns compare 2×H200 and 4×H200 deployments, and curves show serving substrates across target JPS. The gap between effective and evaluated-only quality indicates how much serving failure, timeout, or missing evaluation changes the apparent task outcome.

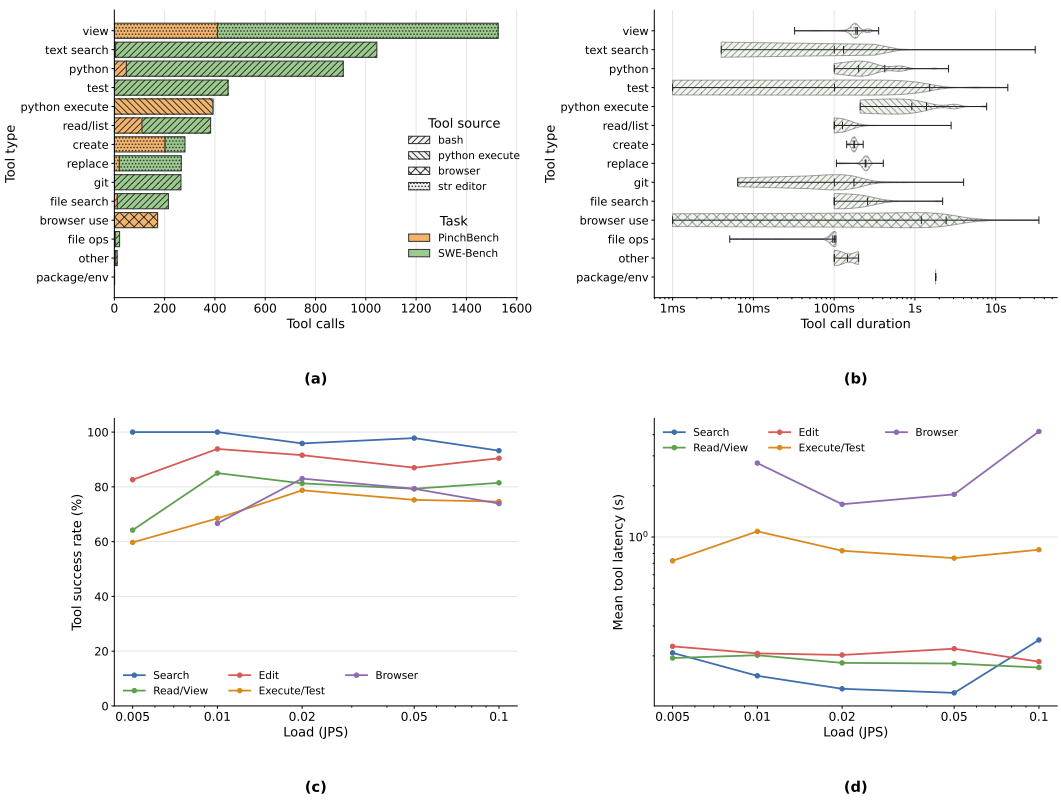


Figure 8: **Tool-layer profile of the mixed MAS workload.** (a) Tool calls are concentrated in read/view, text-search, Python, test, and edit operations, with SWE-Bench contributing most structured-editor and execution-heavy calls and PinchBench contributing browser/search activity. Hatching indicates the tool source. (b) Tool-call durations are highly type-dependent on a logarithmic scale: lightweight read/search/edit calls are short, whereas browser, test, git, and execution-related calls have heavier latency tails. (c) Tool success rates vary by category across load levels, with search remaining the most reliable and browser/edit/execution paths showing lower or more variable success. (d) Mean tool latency also depends strongly on category; browser and execute/test calls dominate tool-time cost and are the most sensitive to high offered load.

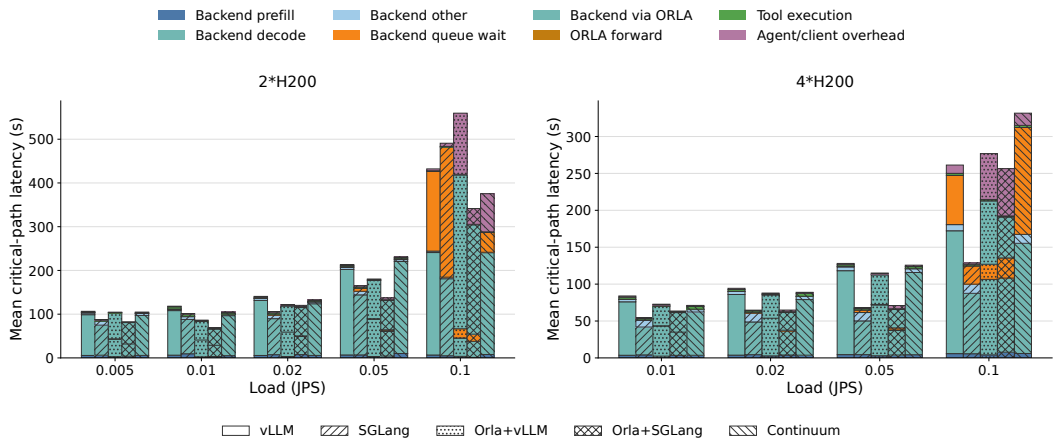


Figure 9: **Runtime decomposition.**