MASEval: Extending Multi-Agent Evaluation from Models to Systems

Cornelius Emde[1,2*], Alexander Rubinstein[3,§], Anmol Goel[1,4,§], Ahmed Heakl[1,5,§], Sangdoo Yun[6], Seong Joon Oh[1,3,7†], Martin Gubri[1‡]
§Equal contribution. [1]Parameter Lab, [2]University of Oxford, [3]University of Tübingen, [4]TU Darmstadt, [5]MBZUAI, [6]NAVER AI Lab [7]KAIST

Abstract

The rapid adoption of LLM-based agentic systems has produced a rich ecosystem of frameworks (smolagents, LangGraph, AutoGen, CAMEL, LlamaIndex, i.a.). Yet existing benchmarks are model-centric: they fix the agentic setup and do not compare other system components. We argue that implementation decisions substantially impact performance, including choices such as topology, orchestration logic, and error handling. MASEval addresses this evaluation gap as a framework-agnostic library that treats the entire system as the unit of analysis. Through the first systematic system-level comparison across 3 benchmarks, 3 models, and 3 frameworks, we find that framework choice matters as much as model choice. MASEval allows researchers to explore all components of agentic systems, opening new avenues for principled system design, and practitioners to identify the best implementation for their use case. MASEval is available under the MIT licence at github.com/parameterlab/MASEval.

1 Introduction

LLM-based agentic systems have become popular for automating complex workflows. Single-agent architectures use one LLM equipped with tools that iteratively reason toward a solution. Yet as workflows grow in complexity, the field is increasingly adopting multi-agent systems, where specialised agents collaborate through structured coordination. This shift promises improved task decomposition and cost efficiency through smaller, specialised models (Belcak et al., 2025). The growing adoption of multi-agent approaches is reflected in the proliferation of frameworks such as AutoGen (Wu et al., 2024), LangGraph (LangChain, 2024a), and CAMEL (Li et al., 2023), which embody different architectural choices for agent communication, memory management, and coordination.

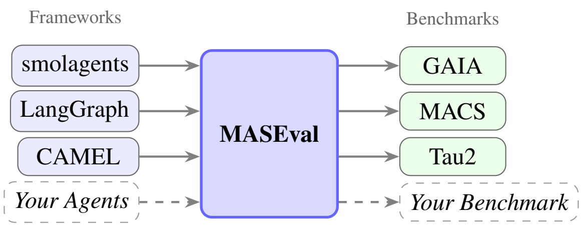

Figure 1: MASEval provides a unified evaluation layer that enables framework-agnostic, system-level comparison across any agent framework and benchmark.

This transition from single-agent to multi-agent systems fundamentally changes evaluation requirements. For single-agent setups, benchmarks could reasonably focus on model capabilities within a fixed agentic scaffold. Multi-agent systems, however, introduce new dimensions that demand attention, including organisational topologies, communication protocols, memory architectures, role differentiation, and the orchestration logic that governs their interaction (Guo et al., 2024). Moreover, the proliferation of frameworks raises a question current benchmarks cannot answer: which framework should I use?

Existing benchmarks such as GAIA (Mialon et al., 2024) and AgentBench (Liu et al., 2024) remain predominantly model-oriented and static. The typical benchmark report states 'GPT-4 achieves 85% on task X,' which conflates model capability with framework implementation. Evaluation libraries for single-agent setups also lack infrastructure to trace multi-agent coordination. This leaves fundamental questions unanswered: is multi-agent topology better than single-agent for task X? Which tool-calling format? Does smolagents outperform LangGraph?

This evaluation gap has concrete consequences. *Researchers* lack a principled way to compare design decisions such as communication topologies

[*]ai@corneliusemde.com
[†]coallaoh@gmail.com
[‡]martin.gubri@parameterlab.de

Library	Multi-Agent	System Eval	Agent-Agnostic	Benchmarks	Flexible Interaction	BYO	Trace-First	Mature
MASEval (Ours)	✓	✓	✓	✓	✓	✓	✓	✓
AnyAgent (Mozilla AI, 2025)	○	✓	✓	✗	○	✓	○	✓
MLflow GenAI (Zaharia et al., 2018)	○	○	✓	✗	○	✓	✓	✓
HAL Harness (Kapoor et al., 2026)	○	✓	✓	✓	○	○	○	○
Inspect-AI (UK AI Safety Institute, 2024)	○	✓	○	✓	○	○	○	✓
OpenCompass (Contributors, 2023)	✗	○	✗	✓	○	○	○	✓
AgentGym (Xi et al., 2024)	✗	✗	✗	✓	○	✓	○	○
Arize Phoenix (Arize AI, 2022)	○	✗	○	✗	✗	○	✓	✓
TruLens (TruEra, 2020)	○	✗	○	✗	✗	○	✓	✓
MARBLE (Zhu et al., 2025)	✓	✗	✗	✓	✗	✗	○	○
DeepEval (Ip and Vongthongsri, 2026)	○	✗	○	✗	○	○	○	✓
MCPEval (LastMile AI, 2025)	✗	✗	✗	✓	✗	○	○	○

Table 1: Comparison with related libraries. ✓ = built-in support or explicitly designed to be extensible, ○ = partial support, ✗ = not supported/impractical. **Multi-Agent**: native orchestration with per-agent tracing and independent message histories. **System-Level**: compare framework implementations, not just LLMs. **Agent-Agnostic**: evaluate any framework via thin adapters without code recreation. **Benchmarks**: ships complete, ready-to-run benchmarks. **Flexible Interaction**: Flexible Agent-Environment-User Interaction. First-class user simulation with personas and tool access. **BYO**: bring your own logging, agents, environments, and tools. Open-source, works offline, no mandatory cloud services. **Trace-First**: evaluate intermediate steps across environment and agents via first-class traces, not post-hoc fixes. **Mature**: published on PyPI, CI/CD, good test coverage, active maintenance.

or coordination strategies and cannot easily build on prior architectural findings. *Practitioners* receive no data-driven guidance on framework choice. *Benchmark consumers* face fragmented interfaces, as evaluating a system across multiple benchmarks requires significant boilerplate reimplementation. *Benchmark producers* must reinvent increasingly complex evaluation infrastructure for each new benchmark.

We introduce MASEval, the first **framework-agnostic evaluation library for multi-agent systems** (Figure 1). Our key principle is to evaluate the complete system (agents, framework, and coordination logic). MASEval provides:

1. **System-level evaluation infrastructure** for comparing design decisions and framework implementations. Both matter for building effective multi-agent systems.
2. **Unified benchmark interface** for evaluating agents across multiple benchmarks with minimal integration overhead.
3. **Benchmark development toolkit** for creating new benchmarks without reinventing evaluation boilerplate.
4. **Multi-agent tracing** with per-agent message histories for debugging coordination patterns.

To validate MASEval's utility, we conduct experiments across 3 benchmarks, 3 frameworks, and 3 models. Our key finding: **framework choice impacts performance comparably to model choice**. This result was previously obscured by existing model-centric benchmarks, and demonstrates the importance of system-level evaluation in MASEval's design. MASEval reduces implementation effort by 83–91% for benchmark consumers who adopt existing benchmarks and by 35–57% for benchmark producers who build new evaluations.

2 Related Work

MASEval is at the intersection of agent frameworks that build systems, benchmarks that define tasks, and evaluation libraries that measure performance.

Multi-agent frameworks. LLM-based agent frameworks span academic systems (Li et al., 2023; Hong et al., 2024; Chen et al., 2024), developer-oriented tools (Wu et al., 2024; LangChain, 2024a; Liu, 2022; CrewAI, 2024; Roucher et al., 2025; Services, 2024), and frontier-provider SDKs (OpenAI, 2025; Google, 2025; Anthropic, 2025). Frameworks differ in design philosophy: stateless vs. stateful execution, static vs. dynamic control flow, centralised vs. decentralised communication, and JSON-based vs. code-based tool calling. AnyAgent (Mozilla AI, 2025) unifies execution across frameworks but does not address evaluation. No prior work provides infrastructure for cross-framework *evaluation* that decouples the system under test from the benchmark harness.

Evaluation libraries. Inspect AI (UK AI Safety Institute, 2024) offers robust sandboxing but collapses multi-agent traces into a single opaque execution, losing per-agent observability. HAL (Kapoor et al., 2026) is limited to single-agent and locks logging to W&B Weave. MARBLE (Zhu et al., 2025) targets multi-agent coordination but requires agents to be reimplemented within its paradigm, precluding cross-framework comparison. LLM observability tools (Arize AI, 2022; TruEra, 2020; Zaharia et al., 2018) focus on monitoring rather than benchmark execution. Commercial platforms (LangChain, 2024b; Braintrust, 2023; Galileo (Rungalileo), 2025) introduce vendor lock-in. None supports evaluating multi-agent systems across frameworks with per-agent tracing. Table 1 compares MASEval with other libraries.

Benchmark datasets. Agent benchmarks cover single-agent capabilities (Jimenez et al., 2024; Mialon et al., 2024; Barres et al., 2025; Liu et al., 2024), multi-agent collaboration (Zhu et al., 2025; Shu et al., 2024; Xu et al., 2024; Froger et al., 2025), and safety (Vijayvargiya et al., 2026; Gomaa et al., 2025). Each ships with an incompatible evaluation interface, requiring bespoke integration code. MASEval's unified benchmark interface addresses this fragmentation.

3 System Architecture

MASEval bridges the gap between frameworks and benchmarks. It is neither an agent framework nor a benchmark dataset, but an *evaluation infrastructure* that enables any agent to be evaluated on any benchmark through a universal interface.

3.1 Design Principles

The challenges from §2 motivate five design principles that distinguish MASEval from prior evaluation approaches:

1. **System as unit of analysis.** The agent system is evaluated as a whole, enabling comparison of architectural choices and framework implementations, not just model capabilities.
2. **Bring your own.** No framework, model provider, or logging backend is privileged. The core never imports framework-specific code (enforced via CI). Pre-built adapters exist for common choices.
3. **Infrastructure, not implementation.** MASEval provides orchestration, tracing, and lifecycle management; users control tools, agent behaviour, and evaluation metrics. Like PyTorch Lightning, it reduces boilerplate without abstracting away evaluation logic.
4. **Separation of concerns.** Task definition (what to solve), environment (tools and state), agent logic (how to solve), and evaluation (how to measure) are cleanly separated. Each component can be varied independently to isolate its effect on performance.
5. **Trace-first evaluation.** All components log to a shared trace organised by component type. Agent messages, model usage, and tool calls can be directly inspected. Per-agent traces are kept independent to be partially observable.

3.2 Module Architecture

MASEval realises these principles through a module structure with strict dependency boundaries that keep evaluation logic independent of any particular framework, model provider, or logging solution (Figure 2).

Core `maseval/core/`. Abstract base classes and evaluation runtime defining the contracts (`AgentAdapter`, `Environment`, `Evaluator`, `ModelAdapter`, `User`). Orchestrates the benchmark lifecycle with minimal dependencies.

Interface `maseval/interface/`. Lazy-loaded adapters for frameworks (smolagents, LangGraph, LlamaIndex), model providers (OpenAI, Anthropic, Google), and logging destinations. Supporting a new framework requires implementing only a thin adapter.

Benchmark `maseval/benchmark/`. Complete benchmark implementations (e.g., τ^2-Bench, MultiAgentBench) serving as both ready-to-use evaluations and reference implementations.

3.3 Core Abstractions

MASEval defines seven core abstractions that benchmark producers implement and benchmark consumers rely on:

Task. The atomic unit of evaluation: bundles the query, environment data, evaluation criteria, and execution protocol.

Benchmark. Orchestrates the evaluation lifecycle for a task collection. Users subclass `Benchmark` and override hooks (`setup_environment`, `setup_agents`, `setup_evaluators`, etc.) while inheriting execution and tracing infrastructure.

Environment. Manages state and exposes tools that agents can invoke, remaining stateful across turns within a task execution.

AgentAdapter. Wraps any framework's agent in a standard interface. Exposes message history for tracing.

User. Simulates user responses for multi-turn benchmarks, with configurable personas and turn limits.

Evaluator. Computes metrics via a two-stage pattern: filter traces to extract relevant data, then compute metrics.

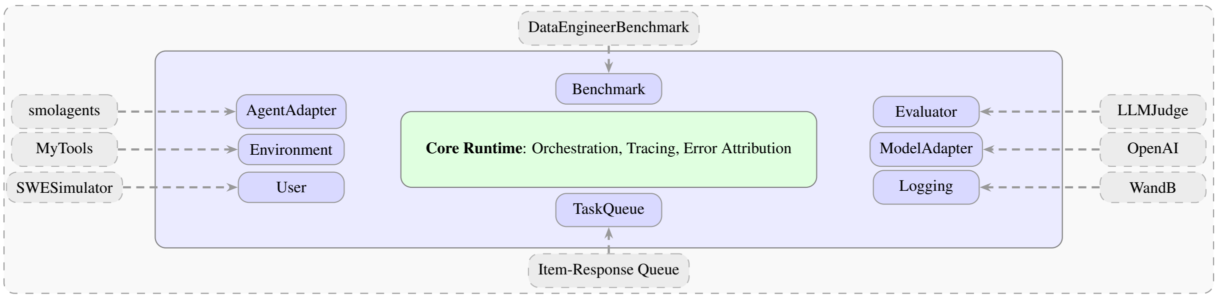

Figure 2: MASEval adopts a "Bring Your Own" (BYO) philosophy. Users implement custom components by extending MASEval's abstract base classes ; the core runtime orchestrates execution and collects traces. This enables maximum flexibility while minimizing boilerplate code.

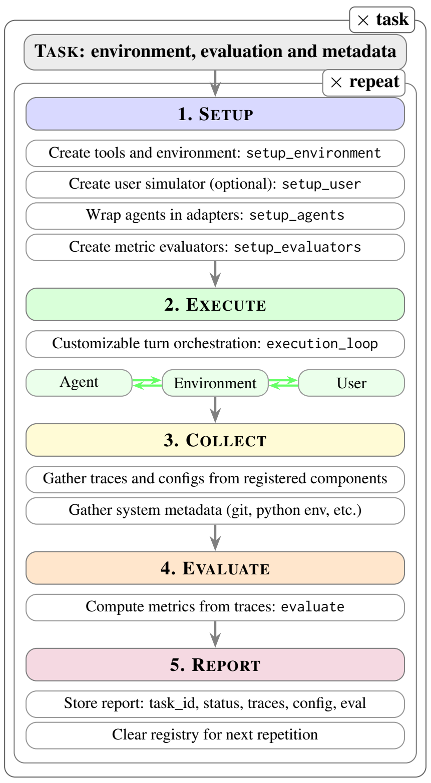
× task
TASK: environment, evaluation and metadata
× repeat
1. SETUP
Create tools and environment: setup_environment
Create user simulator (optional): setup_user
Wrap agents in adapters: setup_agents
Create metric evaluators: setup_evaluators
2. EXECUTE
Customizable turn orchestration: execution_loop
Agent
Environment
User
3. COLLECT
Gather traces and configs from registered components
Gather system metadata (git, python env, etc.)
4. EVALUATE
Compute metrics from traces: evaluate
5. REPORT
Store report: task_id, status, traces, config, eval
Clear registry for next repetition

Figure 3: Benchmark task lifecycle with flexible execution. The outer loop iterates over tasks; the inner loop handles repetitions. The Execute phase shows agent-user interaction as a flexible bidirectional loop.

ModelAdapter. Unified interface for LLM providers with tool calling and token tracking. Used by simulators and LLM-based evaluators; agents use their framework's native integration.

3.4 Benchmark Lifecycle

Each task execution proceeds through five phases (Figure 3), iterated over tasks and repetitions (`n_task_repeats`):

1. **Setup.** Instantiate environment, tools, user simulator, agents, and evaluators; register all components for automatic trace collection.
2. **Execute.** Run agent(s) with customisable turn orchestration; multi-agent coordination happens here.
3. **Collect.** Gather traces from all registered components and system metadata.
4. **Evaluate.** Compute metrics from full traces, including intermediate steps and tool usage.
5. **Report.** Store a structured report (task ID, status, traces, configuration, results) and clear the registry for the next repetition.

3.5 Key Capabilities

Multi-agent tracing. Each agent maintains an independent message history; the component registry automatically collects per-agent traces, enabling debugging of coordination failures.
Callback system. Lifecycle hooks at benchmark, environment, and agent levels enable extensible monitoring without modifying core logic. Built-in callbacks cover progress bars and result logging; users can add custom callbacks for experiment tracking, early stopping, or external platform integration.
Adaptive testing. The `TaskQueue` interface supports custom task selection strategies such as Item Response Theory or DISCO (Rubinstein et al., 2026) to reduce evaluation cost.
Reproducibility infrastructure. Every report includes git state, system information, and package versions; `ConfigurableMixin` extends this to any user-defined component.

3.6 Key Features

MASEval provides production-quality infrastructure for evaluation: parallel task execution, structured error attribution, pluggable logging backends, adaptive task scheduling, and reproducibility tooling. Appendix B has the full list of features. MASEval is installable via `pip install maseval` (MIT licence). The documentation at maseval.readthedocs.io covers API reference, implementation examples and design patterns.

4 System-Level Benchmarking

Current benchmarks focus on model comparison, neglecting *framework implementations* and *design decisions*. We use MASEval to conduct a systematic cross-framework comparison that holds the agent architecture constant and varies only the framework and model. Consider a practitioner who has a multi-agent design in mind and must choose both a model and a framework to implement it. Our experiment measures exactly this: given the same design implemented idiomatically in three frameworks and powered by three models, what performance differences emerge out of the box? We find that framework-level design choices can match or exceed the performance differences between models.

4.1 Experimental Setup

We conduct a full factorial experiment across 3 frameworks, 3 models, and 3 benchmarks (27 configurations). For each benchmark we select 2 domains and run all tasks within each domain.

Benchmarks. We select three benchmarks spanning capability and safety evaluation in multi-agent settings. **MACS** (Shu et al., 2024) tests multi-agent coordination on enterprise tasks. We report partial goal success rate (pGSR). **ConVerse** (Gomaa et al., 2025) measures resistance to contextual privacy attacks in agent-to-agent conversations. We report attack success rate (ASR) and robustness ($1 - \text{ASR}$). The attacker agent is held constant and only the defending agent varies across frameworks. **MultiAgentBench** (Zhu et al., 2025) evaluates both collaboration and competition between agents. We report task completion rate (TCR).

Frameworks. We compare three frameworks through MASEval's built-in adapters. For **smolagents** (Roucher et al., 2025) (minimalist, code-based tool calling) we wrap `ToolCallingAgent` via `SmolAgentAdapter`; for **LangGraph** (LangChain, 2024a) (state machine-based coordination with explicit state management) we wrap `StateGraph` via `LangGraphAgentAdapter`; and for **LlamaIndex** (Liu, 2022) (flexible single- and multi-agent framework) we connect `FunctionAgent` with `AgentWorkflow` via `LlamaIndexAgentAdapter`.

Models. We test three models spanning provider families but remaining within a capability tier for fair comparison: GPT-5-mini, Gemini-3.0-Flash, and Claude-Haiku-4.5. All models are used with temperature 1.0, and `top_p`$= 1.0$ as these are universally available across all models.

Controlled variables. We aim to isolate the design choices made by each framework while keeping all other variables constant. We run each benchmark with hyperparameters chosen to replicate the original paper and applied uniformly across conditions, including tool definitions, agent topologies, user simulations, environment dynamics, evaluation logic, and execution limits (e.g., maximum user turns, step budgets). Where frameworks differ in step granularity (e.g., one step accounting for tool call and response vs. two separate steps), we map limits accordingly. For judges, environment simulation and attackers we utilise the Gemini-3.0-Flash model across all conditions. Only the framework varies, together with its built-in defaults: system prompts, tool-mounting mechanisms, and error handling. We do not modify or align these internals across frameworks.

4.2 Results

Agentic evaluations are inherently noisy due to stochastic model outputs and compounding decision paths, so we focus on aggregate patterns rather than definitive pairwise rankings. Table 2 presents our main results. The bottom rows compare the average range of values across model choice and framework choice in each domain.

Framework impact is substantial. Framework choice produces performance differences comparable in magnitude to model choice. Across the six domains, the range between frameworks exceeds that between models three times. We find the average range between models across the six domains is 11.9 percentage points (pp) and 9.9pp between frameworks. The most striking single-cell example is Haiku 4.5 on MACS Travel, which scores 90.4 with smolagents but 59.5 with LlamaIndex, a 30.9pp gap between frameworks. Practitioners who tune only the model are therefore optimising half the system.

Framework-model interactions. No single framework dominates across all models. On MACS, smolagents achieves the highest scores with Haiku 4.5 but the lowest with GPT-5-mini. To investigate

		MACS		ConVerse		MultiAgentBench	
Framework	**Model**	Travel	Mortgage	Travel Planning	Real Estate	Research	Bargaining
smolagents	Gemini-3.0-Flash	84.0	**94.4**	88.0	82.1	99.0	86.3
	GPT-5-mini	59.8	85.8	90.1	84.2	98.1	**89.6**
	Haiku 4.5	**90.4**	85.6	94.8	92.9	**100.0**	87.7
LangGraph	Gemini-3.0-Flash	85.8	89.4	94.1	91.0	98.0	85.9
	GPT-5-mini	60.8	73.7	80.0	86.4	95.5	79.2
	Haiku 4.5	68.3	81.2	**99.0**	**98.0**	92.4	83.9
LlamaIndex	Gemini-3.0-Flash	74.7	93.2	89.3	96.5	96.0	69.5
	GPT-5-mini	71.0	76.7	90.4	94.2	92.0	88.5
	Haiku 4.5	59.5	76.7	98.6	92.2	95.8	83.9
Range Models		23.6	13.7	11.7	8.9	3.8	9.7
Range Frameworks		17.7	8.7	6.9	10.1	5.6	10.3

Table 2: Performance (% success) across frameworks, models, benchmarks, and domains. Bold is the best per task.

				Reduction	
Benchmark	**Component**	**Original**	**MASEval**	**LoC**	**%**
τ^2-Bench	Definition	6,822	3,450	−3,372	-49.4
	Interface	1,982	343	−1,639	-82.7
	Total	8,804	3,793	−5,011	-56.9
ConVerse	Definition	1,320	1,283	−37	-2.8
	Interface	778	71	−707	-90.9
	Total	2,098	1,354	−744	-35.5

Table 3: Lines of code comparison for benchmark evaluation logic. *Definition* covers task specifications, environments, and evaluators. *Interface* covers the CLI and entry points that wire components into a runnable benchmark. MASEval replaces framework- and benchmark-specific orchestration with shared abstractions.

such strong difference, we carefully analysed traces for smolagents running on MACS and establish surprising results. GPT-5-mini struggles with smolagents as the framework forces a tool call at every step, causing GPT-5-mini to overengage the user through repeated clarification attempts. On MACS, where the number of clarifying questions is capped at five, GPT-5-mini misinterprets the "max turns reached" error message and retries the same tool call up to 23 times, only rephrasing the question rather than adjusting its strategy as response to the systemic failures of this tool call. While it finally can achieve its goals, it often does that with $\geq 10\times$ higher token consumption compared to other models. Other models and frameworks do not exhibit this failure mode. These interactions illustrate how framework conventions (mandatory tool calling, error message format) can combine with model tendencies to produce failures that neither component would exhibit in isolation.

Implications. These results validate MASEval's core premise that framework choice matters for agent performance. Model-only evaluation serves model developers, but practitioners and researchers building agentic systems need more: they must also evaluate the orchestration harness and its interaction with the model. *System-level evaluation infrastructure like MASEval is necessary for the field to make informed architectural decisions.*

4.3 Implementation Effort

Table 3 compares evaluation-related lines of code for MASEval reimplementations against original codebases. Interface code shrinks by 83–91% (the key gain for benchmark *consumers*); overall effort decreases by 35–57% (the relevant figure for benchmark *producers*).

5 Conclusion

MASEval provides framework-agnostic evaluation infrastructure that treats the complete agent system as the unit of analysis. Through multi-agent tracing, structured error attribution, and a unified benchmark interface, it enables systematic comparison of architectural choices and framework implementations. Our experiments show that framework choice impacts performance comparably to model choice, challenging the model-centric status quo. System-level evaluation is what the field needs to move from ad-hoc implementations towards principled system design. We welcome community contributions of additional framework adapters and benchmark integrations.

Ethics Statement

MASEval is evaluation infrastructure that does not introduce new capabilities for language models. The benchmarks we implement were designed by their original authors with appropriate ethical considerations. By lowering the barrier to systematic benchmarking, MASEval could accelerate the development of more capable autonomous agent systems. We believe this risk is outweighed by the benefit of enabling the safety community to identify failure modes and compare mitigation strategies across frameworks in a reproducible manner. We release MASEval under an open-source licence to promote reproducible research and fair comparison in the multi-agent systems community.

Broader Impact Statement

By providing standardised evaluation infrastructure for multi-agent systems, MASEval lowers the barrier to systematic benchmarking of both capability and safety properties. This dual applicability carries inherent tension. On one hand, MASEval can accelerate the development of more effective agentic systems. These systems inherit the risks associated with autonomous AI action, including compounding errors across agents and reduced human oversight in multi-agent coordination. On the other hand, the absence of rigorous, framework-agnostic evaluation tools makes it harder for the safety community to identify failure modes and compare mitigation strategies across implementations. We believe that standardised evaluation is a prerequisite for responsible deployment: systems that cannot be systematically measured cannot be systematically improved.

MASEval currently integrates four frameworks and five benchmarks; broader coverage depends on community contributions. Our architecture has been tested with up to five agents, and scaling behaviour to larger multi-agent systems remains unexplored. Evaluation metrics are task-specific, as each benchmark defines its own success criteria. By design, MASEval prioritises flexibility over convenience. Its lightweight abstractions require more manual work from developers than end-to-end evaluation libraries, a deliberate trade-off to support a wide range of benchmark designs. MASEval's primary audience is the research community, not product development. We expect that benchmark producers and consumers will benefit most from shared evaluation infrastructure that makes safety-relevant findings reproducible and comparable across the fragmented multi-agent landscape.

References

Anthropic. 2025. Claude Agent SDK. https://github.com/anthropics/claude-agent-sdk-python.

Arize AI. 2022. Phoenix: Ai observability & evaluation. https://github.com/Arize-ai/phoenix.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. τ^2-bench: Evaluating conversational agents in a dual-control environment. *Preprint*, arXiv:2506.07982.

Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. 2025. Small language models are the future of agentic ai. *Preprint*, arXiv:2506.02153.

Braintrust. 2023. Braintrust sdk. https://github.com/braintrustdata/braintrust-sdk.

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. 2024. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*.

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

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

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, and 5 others. 2025. Are: Scaling up agent environments and evaluations. *Preprint*, arXiv:2509.17158.

Galileo (Rungalileo). 2025. Galileo-python: Python client library for the galileo platform. https://github.com/rungalileo/galileo-python.

Amr Gomaa, Ahmed Salem, and Sahar Abdelnabi. 2025. Converse: Benchmarking contextual safety in agent-to-agent conversations. *Preprint*, arXiv:2511.05359.

Google. 2025. Agent Development Kit (ADK). https://github.com/google/adk-python. Accessed: 2026-02-07.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pages 8048–8057. International Joint Conferences on Artificial Intelligence Organization. Survey Track.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Jeffrey Ip and Kritin Vongthongsri. 2026. deepeval.

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

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, and 12 others. 2026. Holistic agent leaderboard: The missing infrastructure for AI agent evaluation. In *The Fourteenth International Conference on Learning Representations*.

LangChain. 2024a. LangGraph. `https://github.com/langchain-ai/langgraph`.

LangChain. 2024b. Langsmith: Llm application evaluation and observability platform. `https://docs.smith.langchain.com/`.

LastMile AI. 2025. Mcp-eval: Lightweight eval framework for mcp servers. `https://github.com/lastmile-ai/mcp-eval`. Built on mcp-agent.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. CAMEL: Communicative agents for “mind” exploration of large language model society. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Jerry Liu. 2022. LlamaIndex. Accessed: 2025-12-01.

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, and 3 others. 2024. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. GAIA: a benchmark for general AI assistants. In *The Twelfth International Conference on Learning Representations*.

Mozilla AI. 2025. any-agent. `https://github.com/mozilla-ai/any-agent`.

OpenAI. 2025. OpenAI Agents SDK. `https://github.com/openai/openai-agents-python`.

Aymeric Roucher, Albert Villanova del Moral, Thomas Wolf, Leandro von Werra, and Erik Kaunismäki. 2025. ‘smolagents‘: a smol library to build great agentic systems. `https://github.com/huggingface/smolagents`. Accessed: 2025-12-01.

Alexander Rubinstein, Benjamin Raible, Martin Gubri, and Seong Joon Oh. 2026. DISCO: Diversifying sample condensation for accelerating model evaluation. In *The Fourteenth International Conference on Learning Representations*.

Pydantic Services. 2024. PydanticAI: GenAI Agent Framework. `https://github.com/pydantic/pydantic-ai`.

Raphael Shu, Nilaksh Das, Michelle Yuan, Monica Sunkara, and Yi Zhang. 2024. Towards effective genai multi-agent collaboration: Design and evaluation for enterprise applications. *Preprint*, arXiv:2412.05449.

TruEra. 2020. Trulens: Evaluation and tracking for llm experiments and ai agents. `https://github.com/truera/trulens`.

UK AI Safety Institute. 2024. Inspect ai. `https://github.com/UKGovernmentBEIS/inspect_ai`. Accessed: 2025-12-01.

Sanidhya Vijayvargiya, Aditya Bharat Soni, Xuhui Zhou, Zora Zhiruo Wang, Nouha Dziri, Graham Neubig, and Maarten Sap. 2026. Openagentsafety: A comprehensive framework for evaluating real-world AI agent safety. In *The Fourteenth International Conference on Learning Representations*.

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. 2024. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*.

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. 2024. Agentgym: Evolving large language model-based agents across diverse environments. *Preprint*, arXiv:2406.04151.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, and 2 others. 2024. Theagentcompany: Benchmarking llm agents on consequential real world tasks. *Preprint*, arXiv:2412.14161.

Matei A. 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. 2018. Accelerating the Machine Learning Lifecycle with MLflow. *IEEE Data Eng. Bull.*, 41:39–45.

Yifei Zhou, Song Jiang, Yuandong Tian, Jason Weston, Sergey Levine, Sainbayar Sukhbaatar, and Xian Li. 2025. Sweet-rl: Training multi-turn llm agents on collaborative reasoning tasks. *Preprint*, arXiv:2503.15478.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Robert Tang, Heng Ji, and Jiaxuan You. 2025. 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, Vienna, Austria. Association for Computational Linguistics.

Appendix

A Benchmarks and Frameworks

Tables 4 and 5 report the list of supported benchmarks and agentic frameworks.

Benchmark	Type	Domain
GAIA-2 (Mialon et al., 2024)	Single-agent	Capability
τ^2-bench (Barres et al., 2025)	Single-agent	Capability
MACS (Shu et al., 2024)	Multi-agent	Collaboration
MultiAgentBench (Zhu et al., 2025)	Multi-agent	Coordination
CONVERSE (Gomaa et al., 2025)	Multi-agent	Safety & Security
COLBENCH (Zhou et al., 2025)	Multi-agent	Capability

Table 4: Currently supported benchmarks.

Framework	Architecture
smolagents (Roucher et al., 2025)	Code-based tool calling
LangGraph (LangChain, 2024a)	Stateful graph execution
LlamaIndex (Liu, 2022)	Async-first workflows
CAMEL (Li et al., 2023)	Role-playing multi-agent

Table 5: Currently supported agentic frameworks. Each framework is integrated via a thin adapter that exposes a unified interface for execution, message history, tracing, and configuration capture.

B Key Features

- **Agent framework-agnostic.** Permissive abstract base class with documentation for custom adapters, plus pre-built adapters for smolagents, LangGraph, and LlamaIndex. Adding a new framework requires implementing only two methods: `_run_agent()` and `get_messages()`.
- **Multi-agent native.** Built for multi-agent systems from the ground up with automatic collection of per-agent message histories, maintaining independent conversation contexts that respect partial observability. Each agent sees only its own messages, not other agents' internal states.
- **Comprehensive tracing.** Context-specific trace collection via `TraceableMixin`: agents log steps and message histories; models log input/output pairs, token usage, and latency; tools log invocations with inputs, outputs, and status; simulators log generation attempts and retries. Each component type captures what matters for its role.
- **Benchmark lifecycle management.** The `Benchmark` base class orchestrates a structured execution flow with overridable hooks: `setup_environment()`, `setup_agents()`, `setup_user()`, `setup_evaluators()`, `run_agents()`, and `evaluate()`. Users override specific hooks while inheriting orchestration logic.
- **Callback hook system.** Abstract callback base classes (`BenchmarkCallback`, `EnvironmentCallback`, `AgentCallback`) provide lifecycle hooks at benchmark, environment, and agent levels; built-in callbacks include progress bars (tqdm, Rich) and result logging.
- **Environment abstraction.** Abstract `Environment` base class for custom task environments; users implement `setup_state()` and `create_tools()` to define environment initialization and available tools.
- **Standardized two-stage evaluation.** Abstract `Evaluator` base class with `filter_traces()` to extract relevant data (e.g., specific tool calls) before `__call__()` computes metrics; this separation enables reusable evaluation logic across different trace sources.
- **Custom execution loop.** The `run_agents()` method can be overridden to implement custom turn-taking strategies: user-initiated (user speaks first), model-initiated (agent speaks first), alternating, or fully custom interaction patterns. Default implementation supports configurable multi-turn agent-user exchanges.
- **Multi-turn user simulation.** Abstract `User` base class for custom user simulation logic, plus pre-built `UserLLMSimulator` for LLM-based response generation. Configurable maximum turns, stop tokens for early termination, and automatic conversation history tracking. Supports both message-based turn-taking (standard chatbot interaction) and tool-based interaction for frameworks that model user queries as tool calls (e.g., `ask_user`).
- **Parallel execution.** The `num_workers` parameter enables concurrent task execution via thread pool. All library components are thread-safe: per-thread component registries prevent cross-contamination, locks serialize callback invocations and report aggregation, and trace collection is designed for concurrent access.
- **Structured error attribution.** Exception hierarchy distinguishes `AgentError` (agent vio-

lated contract, i.e., counts against score) from `EnvironmentError` and `UserError` (infrastructure failures, i.e., excluded from scoring). This prevents penalizing agents for benchmark bugs. Developers can raise `AgentError` with an optional `suggestion` field; custom agent implementations can catch these errors and use the suggestion to retry with corrected inputs.

- **Component registry.** Thread-safe registration of agents, models, tools, and simulators via `ComponentRegistry`. Components returned from setup methods are automatically registered; additional components can be manually registered. Enables systematic trace and config collection across all registered components.
- **Unified model interface.** Abstract base class `ModelAdapter` for custom LLM providers, plus pre-built adapters for OpenAI, Google, Anthropic, and HuggingFace with automatic token tracking and tool-calling support.
- **LLM simulators.** Abstract simulator base classes with pre-built `ToolLLMSimulator` and `UserLLMSimulator` that generate realistic tool responses and user turns when real APIs are unavailable, enabling offline development and testing.
- **Configuration snapshotting.** The `ConfigurableMixin` can be added to any user-defined class to participate in configuration capture for reproducibility; built-in collection includes Git state, system information, and package versions.
- **Pluggable logging backends.** Abstract callback base class enables routing results and traces to any destination; JSON file logging is pre-built, with documentation for WandB and Langfuse integration.
- **Debugging tools.** Configurable error handling lets benchmarks continue on failures (`fail_on_task_error=False`) for batch runs or fail fast (`fail_on_task_error=True`) for interactive debugging. Failed tasks are tracked automatically and can be retried selectively.
- **Adaptive testing.** Abstract `TaskQueue` base class enables custom task selection strategies such as Item-Response Theory-based testing or DISCO (Rubinstein et al., 2026) to reduce evaluation costs; priority-based and sequential queues are pre-implemented.
- **Robust task execution with timeout handling and repetition.** The `n_task_repeats` parameter runs each task multiple times for statistical robustness. Reports include `repeat_idx` for aggregation across runs. Per-task deadlines via `TaskContext` with cooperative checkpoint-based timeout (not forced thread termination). Configurable timeout actions: skip, retry, or extend.
- **Structured task protocol.** Each `Task` carries metadata including timeout configuration, retry policies, priority levels, and custom tags for fine-grained control over execution behaviour.