# Process-aware Evaluation with Unified Trace Logging and Robustness Metrics

M.A.C.L Mallikarachchi
Department of Computer Science and Engineering
University of Moratuwa
Email: chemini.21@cse.mrt.ac.lk

Uthayasanker Thayasivam
Department of Computer Science and Engineering
University of Moratuwa
Email: rtuthaya@cse.mrt.ac.lk

***Abstract*—Current benchmarks for evaluating large language model (LLM) agents, such as AgentBench, DeepEval, and IBM's Agentic Evaluation Toolkit, primarily emphasize final outcomes, often overlooking the reasoning pathways, efficiency, and robustness of agent behavior. This paper introduces AutoGen-TraceKit, a lightweight framework designed to capture fine-grained agent interaction traces and derive process-aware and robustness-oriented metrics. Unlike conventional evaluations that assess what agents achieve, AutoGen-TraceKit evaluates how they achieve it by measuring efficiency, stability, control-flow health, and sensitivity to perturbations. The methodology consists of two stages: baseline establishment, where interaction data is systematically recorded and analyzed, and sensitivity analysis, which explores performance under varying configurations and stochastic conditions. Experiments are conducted on the MATH benchmark using the DeepSeek-R1-Distill-Llama-70B model via the Groq API, ensuring both reproducibility and scalability. Results demonstrate that integrating trace logging with structured metrics yields deeper insight into agent strengths, weaknesses, and trade-offs, enabling more transparent and robust evaluations. This approach positions AutoGen-TraceKit as a practical step towards next-generation evaluation frameworks that are reproducible, lightweight, and capable of revealing hidden dynamics in LLM agent reasoning.**

## I. Introduction

The rapid progress of large language models (LLMs) has enabled their deployment as autonomous agents capable of reasoning, problem-solving, and interacting with external environments [1]. Evaluating such agents has become increasingly important, not only to benchmark performance but also to ensure reliability, stability, and trustworthiness in real-world applications. Traditional evaluation frameworks typically focus on whether an agent completes a task successfully, often reducing performance to a single accuracy or success-rate metric. While outcome-based evaluation is simple and comparable, it neglects critical aspects of agent behavior such as reasoning efficiency, error recovery, and robustness under dynamic conditions.

Benchmark-driven approaches such as *AgentBench* [2] provide standardized environments for measuring task success across agents. This enables consistent comparison but largely overlooks how agents arrive at their solutions. In contrast, process-oriented frameworks such as *DeepEval* [3] and IBM's *Agentic Evaluation Toolkit* [4] begin to address this limitation by considering reasoning steps, decision-making, and consistency. However, these frameworks often impose higher integration overheads, lack lightweight traceability, and do not fully capture robustness under stochastic variations. As a result, the field lacks a unified, reproducible, and low-overhead method for capturing both outcome and process-level dynamics in LLM agent evaluation. [5]

In practice, this gap has significant implications. Two agents may achieve the same accuracy score, yet one may do so with far fewer reasoning steps, greater stability under random seeds, and faster recovery from errors. Without process-aware metrics, such differences remain hidden, limiting the ability of researchers and practitioners to assess true agent quality. Furthermore, the absence of robustness testing under varying parameters [6] (e.g., temperature shifts, maximum turn limits) prevents systematic understanding of agent reliability in real-world deployments. This shortcoming is particularly concerning for high-stakes domains such as healthcare, finance, and education, where inefficiencies or unstable reasoning could lead to costly errors, decreased user trust, or safety risks. Thus, evaluation methods that extend beyond task completion to expose reasoning pathways and resilience characteristics are urgently required.

This work introduces *AutoGen-TraceKit*, a lightweight evaluation framework designed to address these shortcomings. The key idea is to extend existing AutoGen pipelines with unified trace logging that captures every interaction step without altering agent logic. From these traces, new process-aware and robustness-oriented metrics are derived, covering efficiency, stability, control-flow health, and resilience under perturbations. Importantly, this approach requires no modifications to the communication protocol or underlying agent architecture, ensuring compatibility with current workflows and negligible computational overhead. In doing so, AutoGen-TraceKit offers a practical, reproducible, and scalable method for advancing evaluation beyond outcome-based benchmarks.

The central research questions guiding this study are as follows. First, can process-aware metrics such as efficiency and stability provide deeper insights into agent behavior compared to outcome-only benchmarks? Second, how sensitive are agent behaviors to stochastic variations in temperature, seeds, and interaction limits? Third, can unified trace logging enable reproducible evaluation across diverse environments while remaining lightweight and scalable? Finally, can such a framework uncover hidden trade-offs in reasoning, such

as the balance between efficiency and accuracy, or stability and adaptability? [7] By addressing these questions, the study aims to advance evaluation practices beyond outcome-based scoring, offering a framework that is transparent, reproducible, and capable of supporting the next generation of robust and trustworthy LLM agents.

The remainder of this paper is organized as follows. Section II presents the related work and background literature relevant to agent evaluation and reasoning assessment. Section III describes the proposed methodology and the design of AutoGen-TraceKit. Section IV discusses the experimental setup and results. Finally, Section V concludes the paper and outlines directions for future work.

## II. Literature Review

The evaluation of large language model (LLM) agents has become a critical area of research as these systems are increasingly deployed in complex reasoning tasks. Traditional evaluation practices primarily emphasize task completion accuracy, providing a binary view of success or failure. While this approach enables straightforward benchmarking, it often neglects the intermediate processes, inefficiencies, and robustness characteristics that define an agent's true capability in dynamic environments.

### A. Benchmark-Oriented Evaluation

Early efforts in evaluating large language model (LLM) agents have primarily focused on outcome-based metrics such as accuracy and task success. A notable example is *AgentBench*, which places agents in simulated environments and measures their ability to achieve predefined goals [2]. Its key advantage lies in standardization, enabling fair comparisons across systems. However, outcome-only evaluation overlooks inefficiencies in reasoning, failure recovery, and adaptability to changes in operating conditions.

### B. Process-Aware Frameworks

Recent frameworks have introduced process-level evaluation to address these shortcomings. *DeepEval*, an open-source tool, facilitates continuous testing of LLM applications by measuring correctness, reliability, and consistency [3]. Similarly, IBM's *Agentic Evaluation Toolkit* emphasizes reasoning quality, decision-making, and error handling, making it valuable for enterprise contexts where stability and trustworthiness are critical [4]. While these frameworks represent progress beyond purely outcome-based methods, they often incur higher integration overheads and lack lightweight, unified trace logging mechanisms.

### C. Limitations of Existing Approaches

Despite advancements, significant gaps remain. Outcome-centric methods fail to capture efficiency trade-offs or highlight wasted reasoning steps. [8] Process-aware methods, while more informative, are not optimized for reproducibility across diverse environments and rarely incorporate robustness analysis under stochastic variations. This limits their ability to expose hidden weaknesses in agent behavior, such as sensitivity to randomness or instability under minor perturbations.

### D. Emerging Directions

The growing recognition of these gaps has motivated research into evaluation frameworks that combine reproducibility, lightweight integration, and richer behavioral metrics. Unified trace logging coupled with sensitivity analysis provides a promising path forward, offering transparency into how agents achieve outcomes rather than focusing solely on whether tasks are completed.

In this context, our work introduces *AutoGen-TraceKit*, which integrates unified trace logging with new process-aware and robustness-oriented metrics. By measuring efficiency, stability, control-flow health, and resilience, it advances current evaluation practices towards more transparent, reproducible, and robust assessments of LLM agents.

## III. Methodology

### Design Principles

The proposed evaluation framework, *AutoGen-TraceKit* [9], is designed to assess not only the correctness of outputs but also the underlying process dynamics of large language model (LLM) agents. Its design is guided by three principles. First, it is non-intrusive, ensuring that the framework passively records agent behavior without influencing the reasoning trajectory or final outputs. Second, it is lightweight, introducing minimal computational overhead so that evaluation results remain unbiased and comparable. Third, it is reproducible, as experiments can be consistently replicated by fixing random seeds and execution parameters, thereby enabling reliable comparisons across runs and configurations.

### Workflow

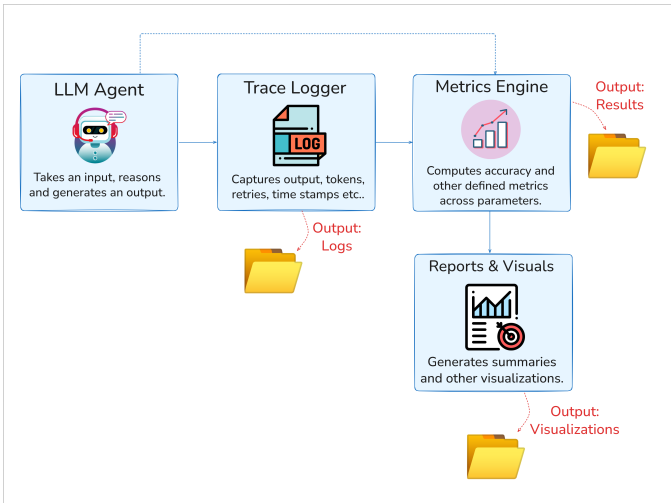



Fig. 1: Workflow of AutoGen Trace-Kit

Figure 1. illustrates the internal workflow of AutoGen-TraceKit. The framework operates in four sequential stages:

agent interaction, trace logging, metric extraction and reporting. During execution, the LLM agent interacts with tasks through the AutoGen pipeline, generating inputs, intermediate steps and final outputs. The Trace Logger passively captures every exchange; inputs, responses, tokens, retries and timestamps without altering the reasoning flow. These structured logs are then processed by the Metrics Engine, which derives quantitative measures such as Token Efficiency (TE) and Control-Flow Health indicators including retry frequency. Finally, these metrics are aggregated into tabular summaries and visual dashboards, enabling transparent, reproducible, and process-aware evaluation of agent performance.

### Trace Logging

Each run produces a structured log that includes the run ID, task ID, model configuration, and execution details. For example:

**Logger Template**

```
Run ID: 20250927_162833_seed42_temp0.7
Problem ID: data/math_easy_int_120.jsonl
Model: llama-3.3-70b-versatile
Seed: 42
Temperature: 0.7
Max Tokens: 1024
Code Execution Enabled: True
Retry Counts: 2
Token Usage: 896
Elapsed Time: 12.4s
Abnormal Termination: False
```

Additional fields capture retry counts, token usage, elapsed time, and abnormal terminations. These logs form the basis for metric computation and later aggregation.

The Methodology can be discussed under two main sections as depicted in

*Stage I: Baseline Evaluation*

In the first stage, the agent is tested under fixed and controlled settings. We keep the same random seed, temperature, and maximum turn limit for every run so that the conditions are stable. During each run, all steps of the agent are carefully logged, including the inputs it receives, the outputs it produces, the number of tokens it uses, and the time taken. From these logs, we calculate both the normal accuracy of the agent's answers and additional process-based measures such as efficiency and stability. The purpose of this stage is to build a clear and reliable baseline of how the agent behaves under standard conditions, which can later be used as a point of comparison for more varied experiments.

*Stage II: Sensitivity Analysis*

In the second stage, we focus on testing how robust the agent is when its conditions are changed. Instead of keeping everything fixed, we run the same tasks while varying important

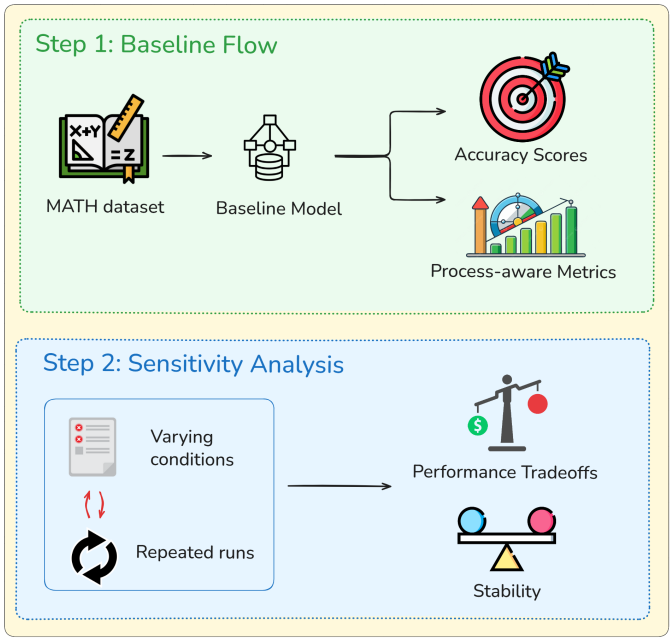



Fig. 2: AutoGen Trace-kit Methodology

parameters such as the random seed, the sampling temperature, and the maximum number of turns allowed. These controlled changes help us see whether the agent can remain stable and consistent when the environment is less predictable. For example, different seeds may lead to small variations in reasoning steps, while higher temperatures may produce more diverse answers. By comparing these results against the baseline from Stage I, we can identify where the agent performs reliably and where it becomes unstable or inefficient. This stage highlights the trade-offs between accuracy, efficiency, and stability when the agent is exposed to uncertainty.

## IV. Experimental Plan

This experimental plan rigorously evaluates the effectiveness of *AutoGen-TraceKit* against outcome-only baselines and existing process-aware tools. The goal is to demonstrate that unified trace logging and process-aware metrics reveal efficiency, stability, control-flow, and robustness characteristics that accuracy alone cannot capture. We structure the plan around a fixed benchmark, a controlled execution setup, two staged conditions (baseline and sensitivity), clear metric definitions, reproducibility practices, and a reporting workflow.

### A. Dataset

We evaluate AutoGen-TraceKit on the **MATH** benchmark [11], restricted to a fixed subset of 120 problems for reproducibility. The dataset is stored in JSONL format, where each line corresponds to a single problem and contains the problem identifier, problem text, ground-truth solution, and final answer. By fixing the dataset to this subset and format, we ensure that results are consistent, reproducible, and interpretable across experimental conditions.

### B. Setup

All experiments are executed as standalone Python scripts. The evaluation uses `DeepSeek-R1-Distill-Llama-70B`, accessed through the Groq API. All configuration parameters including model ID, temperature, maximum tokens, seeds, and turn limits—are defined centrally in `config.py`. This guarantees that runs are fully controlled and reproducible without relying on external configuration files. For each experiment, the framework executes the benchmark tasks, applies the solver, evaluates correctness, and records process metrics. A lightweight trace logger captures metadata (run ID, seed, temperature, timestamp), input/output exchanges, token usage, retries, and termination reasons. Logs are stored in a single location (`experiments/logs/`) using a consistent naming convention (`runID_seed_temp.csv`), ensuring that every run can be uniquely identified and compared. Since all experiments rely on fixed seeds, controlled execution parameters, and a pinned environment, the setup is entirely reproducible across systems.

*Instrumentation (TraceKit)*

The `AutoGen-TraceKit` logger is implemented as a lightweight, non-intrusive module that records every agent–model interaction. It captures not only the exchanged text but also detailed statistics required for efficiency and stability analysis.

**Metrics Captured**

- **Tries** – number of model calls attempted before termination.
- **Format retries** – count of re-prompts triggered due to misformatted outputs.
- **Total tokens** – cumulative tokens consumed across all calls in the run.
- **Walltime (sec)** – total elapsed time measured during execution.
- **Execution successes** – number of successful code executions (when applicable).
- **Final answer** – parsed numerical result extracted from the model's response.
- **Correctness** – Boolean flag indicating whether the final answer matches the gold label.

**Derived Efficiencies**

| Metric | Description |
|---|---|
| **TPS (Tokens per Solve)** | Average tokens spent per successfully solved task. |
| **EPS (Executions per Solve)** | Number of executions required per solved task. |
| **TE (Token Efficiency)** | Solved tasks per 1k tokens consumed. |
| **CE (Call Efficiency)** | Solved tasks per number of LLM calls. |
| **LE (Latency Efficiency)** | Solved tasks per minute of runtime. |

TABLE I: Derived efficiency metrics captured by TraceKit.

**Experimental Conditions**

Stage I: Baseline Evaluation
Runs are conducted with a fixed configuration to establish a reference point. The baseline uses:

- **Seed:** 42
- **Temperature:** 0.7
- **Turn limit:** fixed at 5
- **Max tokens:** 1024

Stage II: Sensitivity Analysis
To measure robustness, we systematically vary random seeds and sampling temperatures while holding other factors constant. Specifically:

- **Seeds:** {42, 123, 999}
- **Temperatures:** {0.5, 0.7, 0.9}
- **Max tokens:** 1024

At the turn level, the logger records raw model inputs and outputs, token usage, and any error or retry signals. When retries are triggered (e.g., due to formatting errors, token-limit issues, or transient API failures), these are appended to the same run record.

Each run file follows a consistent naming convention: `<runID>_seed<seed>_temp<temperature>.csv` ensuring that experiments are uniquely identifiable and directly comparable across conditions.

## V. Discussion

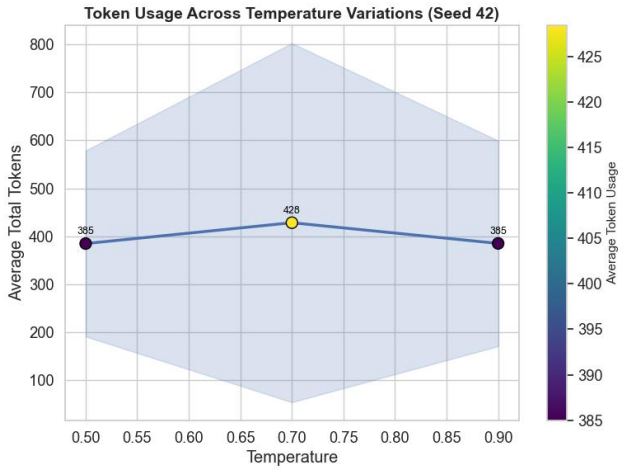
*Stability and Control-Flow Behavior.*

Our experiments highlight the value of process-aware evaluation through `AutoGen-TraceKit`, revealing efficiency and stability trade-offs that would not be visible if we only measured accuracy. [10] Across all runs, final answer correctness remained consistently high (above 96%), suggesting that the model reliably solved the benchmark problems regardless of configuration. However, when we analyze the traces, distinct patterns emerge.

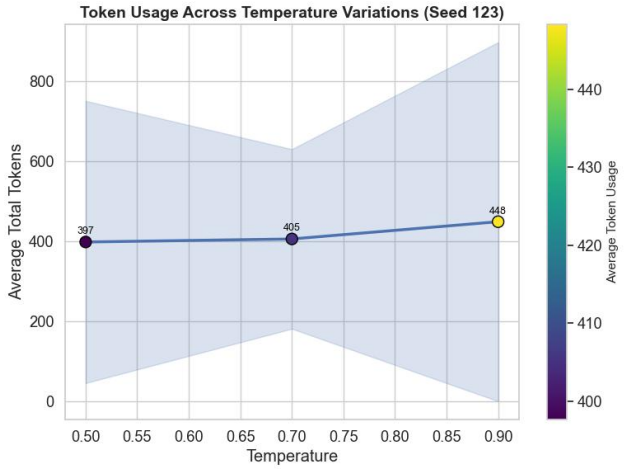
Figures 3a–3c illustrate token usage trajectories across temperatures for individual seeds. While the general trend is stable, some seeds (e.g., 42) show non-monotonic behavior, with higher usage at temperature 0.7 than at either 0.5 or 0.9. This suggests that stability is not only temperature-dependent but also influenced by interaction between seed and sampling randomness. TraceKit's logging of retries and loop indicators further confirms that most runs terminated cleanly, with only occasional format retries. The primary baseline is outcome-only evaluation (accuracy only). This situates TraceKit within the broader evaluation landscape and clarifies its lightweight, unified-logging advantage.

*Efficiency Trends.*

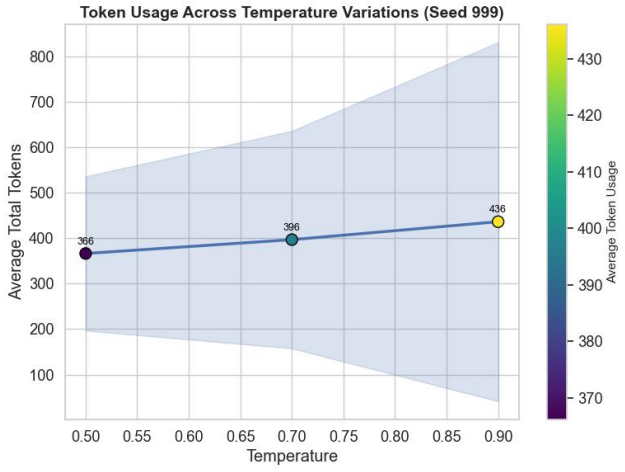
Figure 4 shows that lower temperatures (0.5) achieve better token efficiency, requiring fewer tokens per correct answer compared to higher temperatures (0.7, 0.9). This indicates that while accuracy remains stable, the cost of achieving it increases with higher sampling variability.




(a) SEED-42 Average Token Usage Plot




(b) SEED-123 Average Token Usage Plot




(c) SEED-999 Average Token Usage Plot

Fig. 3: Average token usage across different seeds and temperatures.

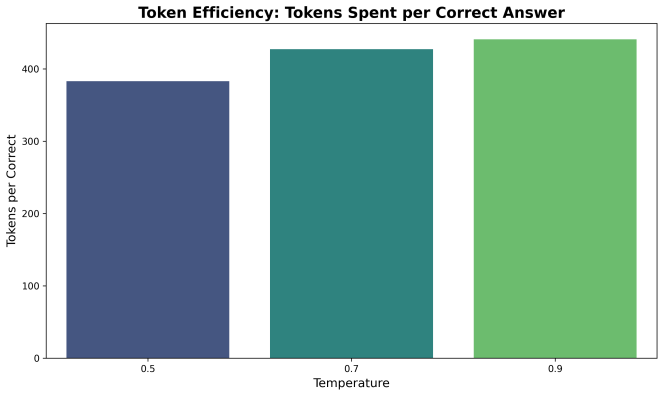



Fig. 4: Token Efficiency

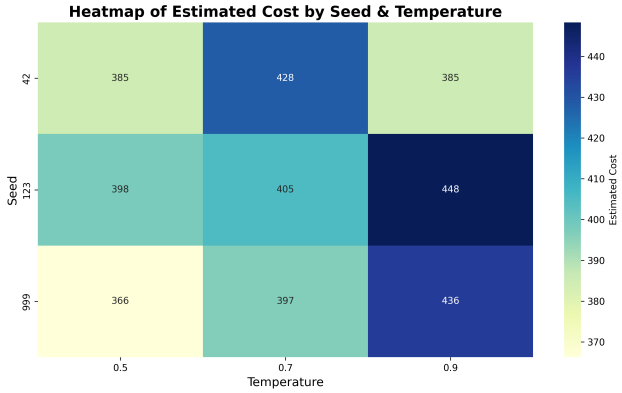



Fig. 5: Heatmap for cost

*Cost Implications*

From a deployment perspective as in Figure 5, efficiency differences translate directly into operational costs. Even small increases in *tokens-per-solve* compound significantly at scale. By linking token usage and wall-clock time into derived metrics such as **TE (Token Efficiency)** and **LE (Latency Efficiency)**, `TraceKit` enables practitioners to quantify these trade-offs in economic terms.

**Value of Process-Aware Evaluation**

Crucially, these findings would not be visible in an accuracy-only evaluation. Standard benchmarks would conclude that all configurations perform equally well, since correctness is consistently high. TraceKit reveals the hidden dimensions of efficiency, stability, and robustness, enabling a more nuanced and realistic understanding of agent performance.

## VI. Conclusion

This work introduced AutoGen-TraceKit, a lightweight and reproducible framework for evaluating large language model agents beyond accuracy. By combining structured trace logging with process-aware metrics, the framework enables a deeper understanding of how agents behave during problem solving. Rather than focusing solely on final correctness, AutoGen-TraceKit surfaces insights into efficiency, stability, control-flow,

and robustness, dimensions that are critical for both research and real-world deployment.

Our experiments on a controlled subset of the MATH benchmark demonstrated the practical value of this approach. While correctness remained consistently high across all tested conditions, process-level metrics revealed substantial differences in token efficiency and cost depending on temperature and seed. The results highlighted efficiency trade-offs, seed-dependent variability, and occasional control-flow instabilities that standard evaluations would have overlooked. By explicitly linking token usage and runtime to derived efficiency metrics, the framework provides an interpretable bridge between model performance and deployment cost.

Looking ahead, AutoGen-TraceKit establishes a foundation for more comprehensive evaluation of LLM agents. Future work may extend the framework to larger and more diverse datasets, integrate latency profiling across hardware backends, and expand control-flow analysis to capture more complex reasoning dynamics. Ultimately, the framework underscores that accuracy is necessary but not sufficient: to responsibly benchmark and deploy agentic systems, process-aware evaluation is indispensable.

## REFERENCES


[1] Q. Wu *et al.*, "AutoGen: Enabling next-gen LLM applications via multi-agent conversation," *https://arxiv.org/abs/2308.08155*, 2023.

[2] Z. Liu *et al.*, "AgentBench: Evaluating LLMs as Agents," *https://arxiv.org/abs/2308.03688*, 2023.

[3] J. Kıcıman *et al.*, "DeepEval: Benchmarking LLMs for reasoning and evaluation," https://github.com/confident-ai/deepeval, 2024.

[4] IBM Research, "Agentic Evaluation Toolkit," 2024. [Online]. Available: https://www.ibm.com/docs/en/watsonx/saas?topic=sdk-agentic-ai-evaluation

[5] Y. Chen *et al.*, "Evaluating Reasoning in Large Language Models," *https://arxiv.org/abs/2404.01869*, 2023.

[6] X. Pan *et al.*, "Robustness Testing of LLMs under Prompt and Parameter Perturbations," *https://arxiv.org/abs/2507.16407*, 2023.

[7] A. Huang *et al.*, "Benchmarking Large Language Models as AI Agents," in *https://www.arxiv.org/pdf/2310.03302v1*, 2023.

[8] A. Yehudai, L. Eden, A. Li, G. Uziel, Y. Zhao, R. Bar-Haim, A. Cohan, and M. Shmueli-Scheuer, "Survey on Evaluation of LLM-based Agents," *arXiv preprint https://arxiv.org/abs/2503.16416*, 2025.

[9] Chemini Mallikarachchi, "AutoGen-TraceKit: Improved approach for evaluating AutoGen math agents by focusing on the reasoning process," *https://github.com/CheliM7/AutoGen-TraceKit*, 2025. [Online; accessed 19 October 2025].

[10] A. Srivastava *et al.*, "Beyond Accuracy: Evaluating Efficiency and Robustness in LLM Agents," *https://arxiv.org/html/2506.11111v2*, 2024.

[11] Hugging Face, "Hendrycks MATH Benchmark Dataset," *https://huggingface.co/datasets/nlile/hendrycks-MATH-benchmark*, 2024.