# RHEA: An Agentic Evaluation and Monitoring Framework

**Pedro Augusto Luiz[1], Wandemberg Gibaut[1], Gabriel Ukstin Talasso[2], Gabriel Paz[3], Thomas da Silva Paula[3], Leandro Villas[2]**

[1]Eldorado Research Institute

[2]University of Campinas (UNICAMP)

[3]HP Inc.

{pedro.augusto,wandemberg.gibaut}@eldorado.org.br,
g235078@dac.unicamp.br, leandro@ic.unicamp.br,
{gabriel.lima.paz1,thomas.paula}@hp.com

***Abstract.*** *We introduce RHEA (Robust Heuristics for Evaluating Agents), a framework-agnostic harness that evaluates LLM workflows under a unified interface, collecting latency, token, and tool-use metrics. On the WorkBench email-and-calendar benchmark, evaluating five LLMs, six frameworks, and two orchestration architectures, we find that model choice has a larger impact on correctness than framework choice (varying 2x-2.5x), while frameworks differ mainly in efficiency. Sequential architectures outperform hierarchical ones in argument correctness (+6.7 pp), while hierarchical architectures reduce token consumption by 34.2%, and parameter mismatch emerges as the dominant failure mode in tool use (39.0%), remaining undetected by coarse task-success metrics.*

## 1. Introduction

Over the past few years, the number of applications for Large Language Models (LLMs) has increased significantly [Kumar 2024], with uses ranging from customer service chatbots to complex decision-making systems. Alongside this growth, there has been a surge in the development of agentic systems, where multiple LLMs or agents interact to solve problems collaboratively [Li et al. 2024].

As these Agentic Systems become more integrated into various sectors, the need for robust evaluation frameworks to assess their performance, reliability, and safety has become even more desirable [He et al. 2024, Yu et al. 2025]. Despite this, existing evaluations of LLM-based multi-agent systems often focus on model-level capabilities, while overlooking how orchestration strategies and tool-use semantics shape the observed performance. In practice, organizations employ diverse agent frameworks (*e.g.*, AgnoAI [Agno 2024], LangGraph [LangChain AI 2024], Microsoft Agents [Microsoft 2024], OpenAI Agents [OpenAI 2025b], PydanticAI [Pydantic 2024], and Smolagents [Hugging Face 2024]), each with distinct control flows, retry logic, and tool adapters. Without a unified protocol, "apples-to-apples" comparisons are elusive, obscuring the trade-offs among accuracy, tool usage, latency, costs and completion stability.

To address this gap, we present the framework RHEA (Robust Heuristics for Evaluating Agents), a harness that encapsulates multiple agent frameworks behind a unified

interface and automatically collects key metrics, including token consumption, time-to-first-response-token (TTFRT), execution time, and tool-use correctness. RHEA supports both *sequential* and *hierarchical* agent architectures and, to balance breadth and tractability, we evaluate agents on the 50-item *WorkBench* email-and-calendar benchmark [Styles et al. 2024], testing five LLMs (GPT-4.1-mini [OpenAI 2025a], GPT-4o-mini [OpenAI 2024], DeepSeek-V3 [Liu et al. 2024], Qwen2.5-72B [Yang et al. 2024], and Mistral-Small-3.2 [Mistral AI Team 2025]) across six agent frameworks under principled controls for model selection and inference parameters.

The main contributions of this paper are:

- **RHEA framework:** a configurable evaluation harness that maps a common agent-workflow specification to heterogeneous frameworks and normalizes their outputs for shared evaluation;
- **Comparative study:** a comprehensive evaluation of widely used agent frameworks under two orchestration architectures and five LLMs, providing evidence-based guidance for framework and model selection.

Our key findings are: (i) model choice is the primary driver of correctness, while framework choice has negligible impact on accuracy but strongly influences efficiency and token consumption; (ii) sequential architectures yield higher argument-level reliability, whereas hierarchical architectures substantially reduce token consumption, and (iii) parameter mismatch is the dominant failure mode and is invisible to coarse tool-selection metrics alone.

The rest of the paper is organized as follows. First, Section 2 reviews related work on multi-agent evaluation and domain-specific benchmarks. Subsequently, Section 3 details the RHEA harness, datasets, and experimental protocol and Section 4 presents quantitative results for hierarchical and sequential agent architectures across email and calendar tasks, covering token consumption, response latency, Tool Selection Accuracy (TSA), and Tool Parameters Accuracy (TPA). Finally, Section 5 concludes the paper with recommendations and directions for future work.

## 2. Related work

### 2.1. Multiagent Systems Evaluation

Multiagent evaluation introduces qualitatively different assessment challenges, requiring frameworks that capture coordination, collaboration, competition, and collective reasoning. Several benchmarks have addressed these dimensions from complementary perspectives. MultiagentBench represents a comprehensive effort to evaluate LLM-based multi-agent systems across diverse interactive scenarios, defining success rate and coordination metrics to assess collaborative and competitive dynamics [Zhu et al. 2025]. LLMArena and BattleAgentBench employ gaming and multi-stage environments to assess capabilities including spatial reasoning, strategic planning, communication, opponent modeling, and team collaboration [Chen et al. 2024, Wang et al. 2024].

The evaluation of collective reasoning represents a particularly sophisticated approach to multi-agent assessment. HiddenBench evaluates whether multi-agent LLM systems can integrate distributed knowledge to reach optimal collective decisions [Li et al. 2025], while Collab-Overcooked introduces process-oriented evaluation metrics to assess fine-grained collaboration capabilities often overlooked by outcome-focused

benchmarks [Sun et al. 2025]. Overall, these studies demonstrate that strong individual-agent capabilities do not necessarily translate into effective multi-agent behavior, with gaps remaining in opponent modeling, team collaboration, collective reasoning, and partner-awareness. These findings suggest that emergent multi-agent capabilities require explicit architectural, training, and evaluation considerations.

### 2.2. Benchmarking Datasets

Domain-specific benchmarks have proliferated, recognizing that general-purpose evaluation may not adequately capture the specialized knowledge, workflows, and constraints of particular application areas. Frameworks like BLADE [Gu et al. 2024], DataSciBench [Zhang et al. 2025], CRMArena-Pro [Huang et al. 2025], and PyBench [Zhang et al. 2024] provide targeted assessment for data science, business processes, and software development.

The most common dataset to evaluate LLMs' capabilities is the Massive Multitask Language Understanding (MMLU) [Hendrycks et al. 2020] that contains a large, diverse collection of multiple-choice questions covering 57 academic and professional subjects, designed to measure a model's factual knowledge. While highly scalable and statistically robust, this formulation evaluates models as passive solvers of isolated problems and must be integrated with other tools and components to evaluate complex LLM agent systems.

More recent benchmarks move toward interactive and execution-grounded evaluation. SWE-bench [Jimenez et al. 2023] evaluates real-world software maintenance, where agents must modify large GitHub repositories so that failing tests pass, introducing environment interaction, long context reasoning, and tool-like behavior. DevAI [Zhuge et al. 2024] extends this direction by modeling each task as a structured development process composed of hierarchical requirements, enabling fine-grained trajectory-level-assessment of agent behavior. WorkBench [Styles et al. 2024] takes a different angle, providing an open-source benchmark for evaluating agents on realistic workplace tasks. It contains five sandbox databases covering email, calendar, analytics, customer relationship management, and project management, along with different read/write tools. In this work we use the multi-domain email and calendar subset of WorkBench, comprising 50 tasks that require agents to schedule meetings, compose emails, and manage calendar events under conditional logic. This execution-grounded design makes it particularly suited for our study, and we adopt it as our evaluation benchmark to systematically compare agent frameworks, LLMs, and orchestration architectures.

## 3. Methodology and Experiments

To ensure fair and reproducible comparisons across agent frameworks, this work introduces RHEA, a standardized evaluation harness packaged as a Python toolkit that encapsulates AgnoAI, LangGraph, Microsoft Agents, OpenAI Agents, PydanticAI, and Smolagents behind a unified interface. RHEA uses a common YAML specification for task inputs, agent configurations, and tools, which framework-specific adapters translate into native workflows. The resulting executions are normalized into a common format for consistent metric computation, with metrics automatically collected at each run without relying on external monitoring tools.

### 3.1. Datasets

To conduct the experiments, we use the *WorkBench* email and calendar benchmark [Styles et al. 2024], a 50-task suite designed to evaluate agents on personal assistant workflows. Each task is presented in natural language and requires the agent to first read and reason over a simulated inbox and calendar and then perform zero or more write side actions, such as send an email or create an event conditioned on the observed state. Additionally, thirteen of the 50 tasks are *conditional negatives*, representing scenarios where the triggering condition is not met and the correct response is to take no write action.

Each task is annotated with a ground-truth, a list of expected tool invocations together with their required argument values (e.g., recipient address, event start time, duration). This annotation allows for fine-grained evaluation of tool-use correctness at both the tool-name and parameter levels, enabling us to distinguish between cases where the agent calls the correct tool but with incorrect arguments (parameter mismatch) and cases where the agent fails to call the required tool at all (tool-name error).

### 3.2. Architectures

Two orchestration patterns are evaluated for the email and calendar use case, as illustrated in Figure 1, with the same reader and writer prompts, tool access, and model configurations across both designs.

In the hierarchical architecture, a manager agent receives the user request and delegates subtasks to specialized sub-agents. A reader sub-agent is responsible for information retrieval and condition evaluation, while a writer sub-agent executes the appropriate write-side actions based on the manager's instructions. The manager performs no task-specific tool calls and serves solely as a coordination layer, introducing an additional delegation and context-relaying step.

In the sequential architecture, two agents operate in a linear pipeline: a reader agent first queries the read-side tools (inbox search, email retrieval, calendar lookup) to gather the necessary context and evaluate conditional logic, and a downstream writer agent consumes the reader's output to issue the appropriate write-side tool calls. Execution follows a strict step-by-step control flow, with no separate manager or delegation layer, agents communicate only through the sequentially passed context.

### 3.3. Evaluation Metrics

To evaluate our experiments, we define a set of metrics capturing both correctness and efficiency aspects of agent performance. The primary correctness metrics are Tool Selection Accuracy (TSA) and Tool Parameters Accuracy (TPA), defined as:

$$\text{TSA} = \frac{N_{\text{correct tools}}}{N}, \qquad \text{TPA} = \frac{N_{\text{correct args}}}{N} \tag{1}$$

where $N$ is the total number of runs, $N_{\text{correct tools}}$ is the number of runs whose predicted write-tool names exactly match the ground-truth multiset, and $N_{\text{correct args}}$ is the number of runs where all corresponding arguments also match. TSA and TPA measure whether the required tool(s) are invoked and whether all arguments in the tool calls match the ground truth, respectively. A run is classified as PASS only when the agent invokes exactly the expected write tool(s) with all arguments matching the ground truth. Any deviation, including incorrect arguments, missing required calls, spurious write actions in

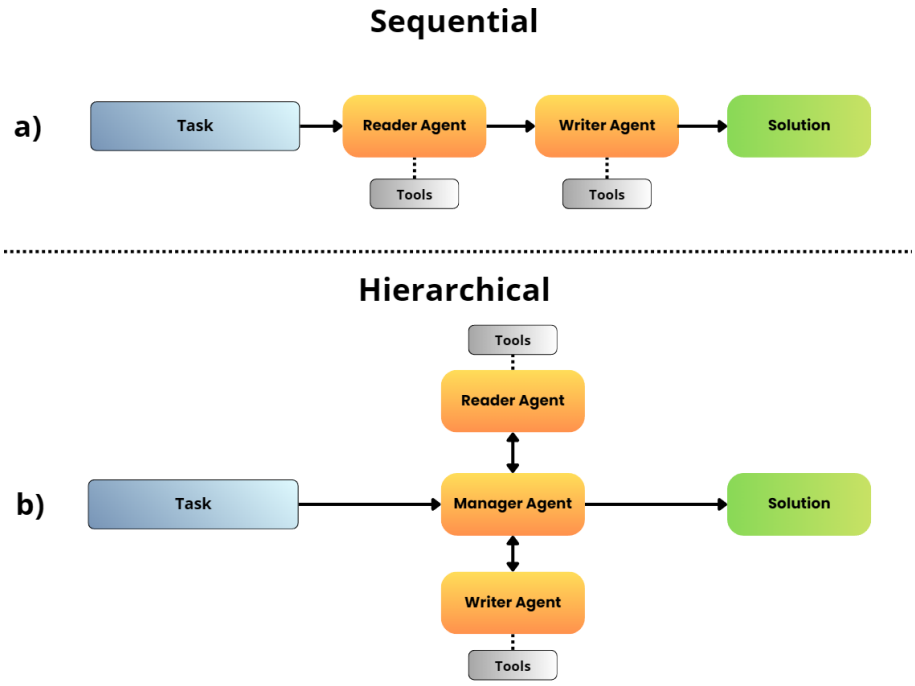



**Figure 1. Sequential and Hierarchical architectures for the email and calendar task. In the sequential approach (a), a reader agent gathers inbox and calendar state and evaluates the condition, and a writer agent then issues the required write side tool calls. In the hierarchical approach (b), a manager agent delegates information retrieval to a reader sub-agent and write side execution to a writer sub-agent.**

tasks that require none, or extra calls, is classified as FAIL. In parallel, efficiency is measured through total token consumption, end-to-end execution time from task invocation to workflow completion, time to first response token (TTFRT) from task invocation to the first streamed token, and throughput (tokens per second). By combining correctness and efficiency metrics, we analyze the trade-offs across frameworks and architectures (see Section 4.2).

### 3.4. Experimental Setup

All experiments were conducted via the OpenRouter API [1] for open-source models and OpenAI API [2] for OpenAI models using the same configuration (Intel Core i7-13700 and 32GB RAM). All inference parameters were set to each framework's default values, and each of the 50 WorkBench tasks was executed by a combination of model, framework, and architecture, yielding a total of $50 \times 5 \times 6 \times 2 = 3{,}000$ task runs.

## 4. Results

As described in Section 3, using RHEA on the WorkBench email and calendar benchmark, we compare agent frameworks, LLMs, and orchestration architectures across Tool Selection Accuracy (TSA), Tool Parameters Accuracy (TPA), token consumption, and latency. Analyses are computed over 3,000 task runs covering 5 models, 6 frameworks, 2 architectures, and 50 tasks per combination.

[1] https://openrouter.ai/
[2] https://openai.com/api/

### 4.1. Task Effectiveness

We evaluate effectiveness at two granularities: (i) *Tool Selection Accuracy (TSA)*, which checks whether the correct write tools were called by name, and (ii) *Tool Parameters Accuracy (TPA)*, which verifies whether every argument of each tool call matches the ground-truth value exactly. As shown in Table 1 and Figure 2, the two metrics diverge substantially.

First, at the model level, DeepSeek-V3 achieves the highest TPA (0.390), followed by GPT-4.1-mini (0.340). In contrast, TSA is similar across most models and does not discriminate between them as clearly. For instance, GPT-4o-mini reaches a TSA of 0.552 but only 0.270 in TPA, while Mistral-Small-3.2 underperforms on both dimensions (TSA 0.383, TPA 0.152), confirming that coarse success metrics significantly overstate actual reliability.

At the framework level (Table 2), TPA is remarkably uniform across five of the six frameworks, all at 0.286, except for Smolagents, which reaches 0.302 but at substantially higher resource cost (Section 4.2). This parity makes efficiency the primary differentiator for framework selection in this benchmark.

Thus, in tool-calling agents deployed in production, where incorrect argument values cause real downstream failures, coarse success rates are misleading proxies for reliability. Additionally, TPA varies strongly across models (0.152→0.39, +156%) but slightly across frameworks (0.286→0.302, +5.6%), whereas TSA shows similar variation across both models and frameworks (0.383→0.552, +44% vs. 0.402→0.572, +42%). This suggests that TPA is more sensitive to model choice than to framework selection.

**Table 1. Aggregate results per LLM (600 runs each, all frameworks and architectures). Best estimates or means per column in bold.**

| Model | $n$ | TSA | TPA | Tokens | Time (s) | TTFRT (s) |
|---|---|---|---|---|---|---|
| **DeepSeek-V3** | 600 | 0.550 [0.510, 0.589] | **0.390** [0.352, 0.430] | 6,216 ± 5,148 | 25.94 ± 17.68 | 15.03 ± 16.35 |
| GPT-4.1-mini | 600 | 0.517 [0.477, 0.556] | 0.340 [0.303, 0.379] | **4,187** ± 2,892 | **8.53** ± 5.67 | **6.12** ± 4.78 |
| Qwen2.5-72B | 600 | 0.542 [0.502, 0.581] | 0.292 [0.257, 0.329] | 8,483 ± 10,652 | 33.93 ± 88.99 | 28.48 ± 86.32 |
| GPT-4o-mini | 600 | **0.552** [0.512, 0.591] | 0.270 [0.236, 0.307] | 5,067 ± 4,829 | 10.32 ± 5.54 | 7.39 ± 4.05 |
| Mistral-Small-3.2 | 600 | 0.383 [0.345, 0.423] | 0.152 [0.125, 0.183] | 7,010 ± 13,252 | 12.44 ± 20.72 | 8.51 ± 16.20 |

**Table 2. Aggregate results per agent framework (500 runs each, all models and architectures). Best estimates or means per column in bold.**

| Framework | $n$ | TSA | TPA | Tokens | Time (s) | TTFRT (s) |
|---|---|---|---|---|---|---|
| AgnoAI | 500 | **0.572** [0.528, 0.615] | 0.286 [0.248, 0.327] | 4,938 ± 4,208 | 21.16 ± 18.08 | 16.05 ± 16.36 |
| MS Agents | 500 | 0.564 [0.520, 0.607] | 0.286 [0.248, 0.327] | 3,735 ± 2,280 | 12.73 ± 9.06 | **8.41** ± 7.21 |
| OpenAI Agents | 500 | 0.538 [0.494, 0.581] | 0.286 [0.248, 0.327] | 5,820 ± 4,827 | 12.92 ± 9.28 | 9.45 ± 7.50 |
| LangGraph | 500 | 0.502 [0.458, 0.546] | 0.286 [0.248, 0.327] | 7,908 ± 16,759 | 18.89 ± 26.54 | 17.57 ± 26.02 |
| PydanticAI | 500 | 0.474 [0.431, 0.518] | 0.286 [0.248, 0.327] | **3,683** ± 2,912 | **9.80** ± 7.70 | 8.51 ± 7.14 |
| Smolagents | 500 | 0.402 [0.360, 0.446] | **0.302** [0.263, 0.344] | 11,234 ± 7,074 | 34.64 ± 98.55 | 18.92 ± 95.43 |

### 4.2. Token Consumption and Latency

Figure 3 shows mean token consumption per combination, Figure 4 shows the execution time distribution, and Figure 5 shows the efficiency trade-off between parameter correctness and mean token usage.

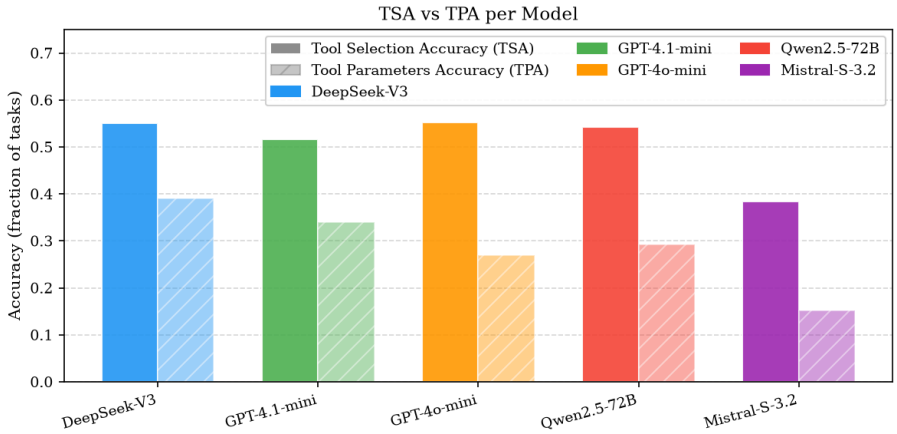



**Figure 2. TSA vs. TPA per model: Tool Selection Accuracy (solid bars) versus Tool Parameters Accuracy (hatched bars). The gap between the two metrics reveals how often models call the right tool but with incorrect arguments.**

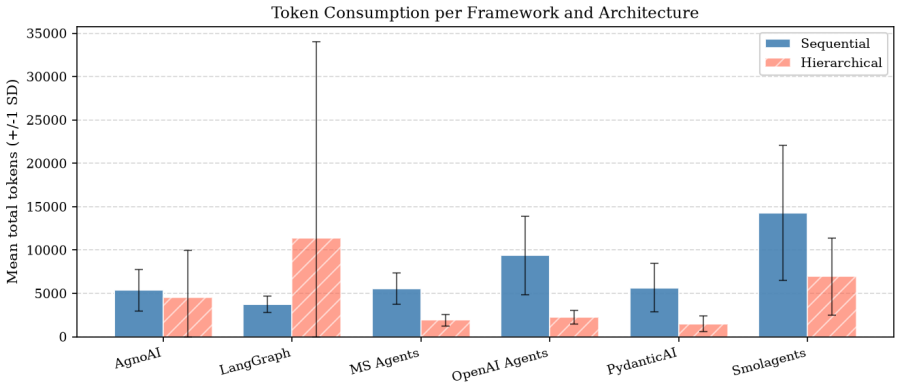



**Figure 3. Mean total tokens per task per framework, separated by architecture (sequential and hierarchical).**

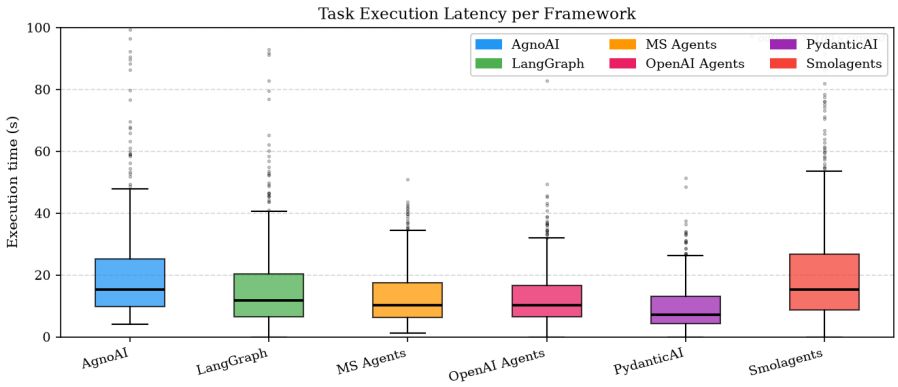



**Figure 4. Execution time distribution per framework (both architectures combined). Outliers above 100 s are clipped.**

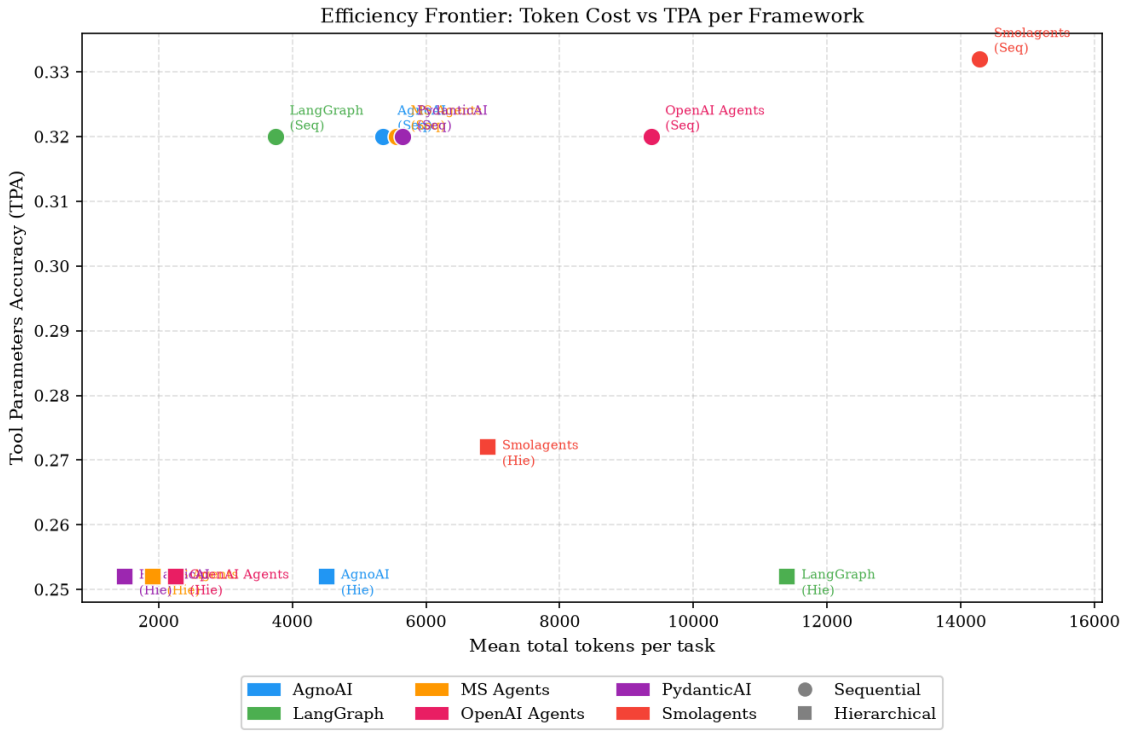



**Figure 5. Efficiency frontier: mean token cost vs. TPA per framework and architecture. Points closer to the top-left corner are preferable (high TPA, low cost). Circles = sequential, squares = hierarchical.**

As observed, MS Agents and PydanticAI are the most token-efficient frameworks (3,735 and 3,683 tokens respectively) with competitive latency (12.73s and 9.80s), while Smolagents consumes more than three times as many tokens as PydanticAI (11,234 vs 3,683) and is the slowest framework in both total time (34.64s) and TTFRT (18.92s).

At the model level, GPT-4.1-mini is the most latency-efficient (8.53s, TTFRT 6.12s) while also achieving the second-highest parameter correctness (0.340). In contrast, DeepSeek-V3 achieves the highest correctness but at nearly three times the latency (25.94s), whereas Qwen2.5-72B exhibits the highest token consumption (8,483) and the longest end-to-end time (33.93s, TTFRT 28.48s), indicating substantial overhead relative to its correctness score.

### 4.3. Effect of Orchestration Architecture

To analyze differences between the two orchestration designs, we evaluate 1,500 matched (model, framework, task) triples using exact two-sided McNemar tests and paired-bootstrap 95% confidence intervals (CIs). Results are summarized below:

- **TPA:** sequential 0.322 vs. hierarchical 0.255 ($\Delta = +0.067$, 95% CI $[0.039, 0.095]$, $p < 0.001$); sequential architectures achieve significantly higher argument correctness.
- **TSA:** sequential 0.504 vs. hierarchical 0.513 ($\Delta = -0.009$, 95% CI $[-0.039, 0.021]$, $p = 0.583$); the difference is not statistically significant.
- **Token consumption:** sequential 7,437 vs. hierarchical 4,891 ($-34.2\%$). Hierarchical delegation substantially reduces total tokens.

- **Execution time:** sequential 17.00s vs. hierarchical 19.50s ($\Delta = +2.50$s). Despite fewer tokens, hierarchical execution is marginally slower due to multi-agent coordination overhead.

### 4.4. Failure Analysis

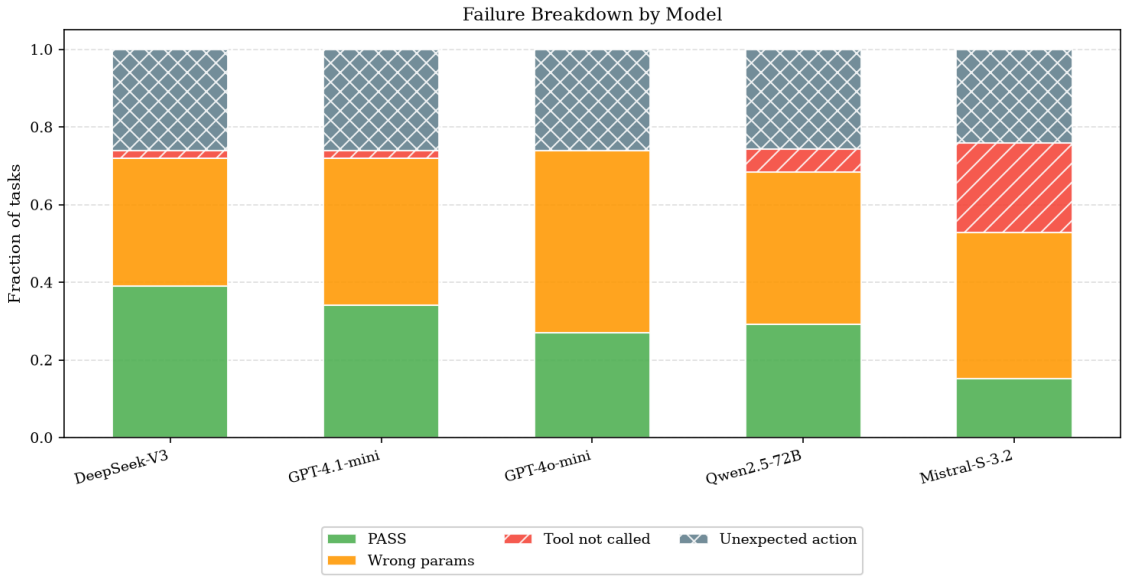



**Figure 6. Failure breakdown per model: stacked bars show the fraction of runs classified as PASS, wrong parameters, tool not called, or unexpected write action. Results averaged over all frameworks and architectures.**

As observed, Figure 6 illustrates failures across all 3,000 runs. Of these, 28.9% are full successes (PASS). The dominant failure mode is *parameter mismatch* (39.0%): the correct tool was called but with at least one wrong argument value (e.g., incorrect event time or wrong recipient address). This is the failure type invisible to coarse task-success metrics.

The second most frequent failure mode is *unexpected write action* (25.6% of all runs). TSA was higher on the 37 tasks requiring a write action (0.575, 95% CI [0.555, 0.596]) than on the 13 conditional-negative tasks requiring abstention (0.319, 95% CI [0.287, 0.353]), revealing a weakness in conditional abstention. Finally, *tool not called* accounts for 6.6% of runs, corresponding to cases where the agent completes the task without issuing the required write action.

These results suggest that parameter mismatch failures may be mitigated at the framework level through structured output schemas or argument validation mechanisms, whereas conditional-negative failures point to a reasoning limitation in the models, potentially requiring more explicit condition-checking prompts prior to tool execution, particularly in scenarios where those conditions are already specified in the initial instructions prompts.

### 4.5. Summary of Findings

- TPA is a substantially more discriminative metric than TSA and reveals differences hidden by macro evaluation.

- DeepSeek-V3 achieves the highest TPA (0.390), and GPT-4.1-mini offers the best TPA-to-latency trade-off (TPA 0.340, 8.53s).
- Sequential orchestration improves TPA (+6.7 pp), and hierarchical orchestration reduces token consumption ($-34.2\%$) at the cost of a small latency increase.
- Among frameworks, MS Agents and PydanticAI are the most token-efficient, and AgnoAI achieves the highest task success rate.
- The dominant failure mode is parameter mismatch (39.0%); TSA drops from 0.575 on positive tasks to 0.319 on conditional-negative tasks.

## 5. Conclusion

This paper presents RHEA, an evaluation harness for agentic LLM frameworks, and reports results from 3,000 task runs across five LLMs, six frameworks, and two orchestration architectures on the WorkBench email and calendar benchmark. Our results show that model choice is the primary determinant of correctness, as reflected by TPA ranging from 0.152 to 0.390 and TSA from 0.383 to 0.552 across models, while framework choice produces negligible differences (0.286 to 0.302) and mainly affects efficiency, such as token consumption and latency. Sequential architectures yield higher argument-level correctness (TPA +6.7 pp), whereas hierarchical architectures reduce token consumption by 34.2%, and among frameworks, MS Agents and PydanticAI offer the best efficiency trade-off, while DeepSeek-V3 and GPT-4.1-mini lead in TPA and latency, respectively. We further find that coarse task-success metrics such as Tool Selection Accuracy (TSA) are insufficient to capture the dominant failure mode, parameter mismatch, which accounts for 39.0% of all runs and is not observable through TSA alone. This highlights the limitation of coarse task-success metrics and the need to consider parameter-level correctness in agent evaluation.

In future work, we will extend RHEA to additional task domains, architectures, and a wider range of models to examine whether these trade-offs persist in more complex, long-horizon settings, where hierarchical coordination may better support task decomposition and specialized agents.

## References


Agno (2024). Agno: The complete stack for AI agents – framework, runtime and control plane. `https://github.com/agno-agi/agno`.

Chen, J., Hu, X., Liu, S., Huang, S., Tu, W.-W., He, Z., and Wen, L. (2024). Llmarena: Assessing capabilities of large language models in dynamic multi-agent environments. *arXiv preprint arXiv:2402.16499*.

Gu, K., Shang, R., Jiang, R., Kuang, K., Lin, R.-J., Lyu, D., Mao, Y., Pan, Y., Wu, T., Yu, J., et al. (2024). Blade: Benchmarking language model agents for data-driven science. *arXiv preprint arXiv:2408.09667*.

He, F., Zhu, T., Ye, D., Liu, B., Zhou, W., and Yu, P. (2024). The emerged security and privacy of llm agent: A survey with case studies. *ACM Computing Surveys*.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. (2020). Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Huang, K.-H., Prabhakar, A., Thorat, O., Agarwal, D., Choubey, P. K., Mao, Y., Savarese, S., Xiong, C., and Wu, C.-S. (2025). Crmarena-pro: Holistic assessment of llm agents across diverse business scenarios and interactions. *arXiv preprint arXiv:2505.18878*.

Hugging Face (2024). smolagents: A barebones library for agents that think in code. `https://github.com/huggingface/smolagents`.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. (2023). Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

Kumar, P. (2024). Large language models (llms): survey, technical frameworks, and future challenges. *Artificial Intelligence Review*, 57(10):260.

LangChain AI (2024). LangGraph: Build resilient language agents as graphs. `https://github.com/langchain-ai/langgraph`.

Li, X., Wang, S., Zeng, S., Wu, Y., and Yang, Y. (2024). A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1):9.

Li, Y., Naito, A., and Shirado, H. (2025). Assessing collective reasoning in multi-agent llms via hidden profile tasks. *arXiv preprint arXiv:2505.11556*.

Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al. (2024). Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Microsoft (2024). AutoGen: A framework for building, orchestrating and deploying AI agents and multi-agent workflows. `https://github.com/microsoft/autogen`.

Mistral AI Team (2025). Mistral small 3.2 (24b): Model card and repository. Hugging Face. Accessed: 2026-05-22.

OpenAI (2024). Gpt-4o mini: Advancing cost-efficient intelligence. Accessed: 2024-11-21.

OpenAI (2025a). Introducing GPT-4.1 in the API. `https://openai.com/index/gpt-4-1/`.

OpenAI (2025b). OpenAI Agents SDK: A lightweight, powerful framework for multi-agent workflows. `https://github.com/openai/openai-agents-python`.

Pydantic (2024). PydanticAI: Genai agent framework, the Pydantic way. `https://github.com/pydantic/pydantic-ai`.

Styles, O., Miller, S., Cerda-Mardini, P., Guha, T., Sanchez, V., and Vidgen, B. (2024). Workbench: a benchmark dataset for agents in a realistic workplace setting.

Sun, H., Zhang, S., Niu, L., Ren, L., Xu, H., Fu, H., Zhao, F., Yuan, C., and Wang, X. (2025). Collab-overcooked: Benchmarking and evaluating large language models as collaborative agents. *arXiv preprint arXiv:2502.20073*.

Wang, W., Zhang, D., Feng, T., Wang, B., and Tang, J. (2024). Battleagentbench: A benchmark for evaluating cooperation and competition capabilities of language models in multi-agent systems. *arXiv preprint arXiv:2408.15971*.

Yang, A., Yang, B., Hui, B., Zheng, B., Yu, B., Zhou, C., Li, C., Li, C., Liu, D., Huang, F., Dong, G., Wei, H., Lin, H., Tang, J., Wang, J., Yang, J., Tu, J., Zhang, J., Ma, J., Yang, J., Xu, J., Zhou, J., Bai, J., He, J., Lin, J., Dang, K., Lu, K., Chen, K., Yang, K., Li, M., Xue, M., Ni, N., Zhang, P., Wang, P., Peng, R., Men, R., Gao, R., Lin, R., Wang, S., Bai, S., Tan, S., Zhu, T., Li, T., Liu, T., Ge, W., Deng, X., Zhou, X., Ren, X., Zhang, X., Wei, X., Ren, X., Liu, X., Fan, Y., Yao, Y., Zhang, Y., Wan, Y., Chu, Y., Liu, Y., Cui, Z., Zhang, Z., Guo, Z., and Fan, Z. (2024). Qwen2 technical report.

Yu, M., Meng, F., Zhou, X., Wang, S., Mao, J., Pan, L., Chen, T., Wang, K., Li, X., Zhang, Y., et al. (2025). A survey on trustworthy llm agents: Threats and countermeasures. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 6216–6226.

Zhang, D., Zhoubian, S., Cai, M., Li, F., Yang, L., Wang, W., Dong, T., Hu, Z., Tang, J., and Yue, Y. (2025). Datascibench: An llm agent benchmark for data science. *arXiv preprint arXiv:2502.13897*.

Zhang, Y., Pan, Y., Wang, Y., and Cai, J. (2024). Pybench: Evaluating llm agent on various real-world coding tasks. *arXiv preprint arXiv:2407.16732*.

Zhu, K., Du, H., Hong, Z., Yang, X., Guo, S., Wang, D. Z., Wang, Z., Qian, C., Tang, R., Ji, H., et al. (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.

Zhuge, M., Zhao, C., Ashley, D., Wang, W., Khizbullin, D., Xiong, Y., Liu, Z., Chang, E., Krishnamoorthi, R., Tian, Y., et al. (2024). Agent-as-a-judge: Evaluate agents with agents. *arXiv preprint arXiv:2410.10934*.