# AgentArch: A Benchmark for Evaluating Agent Architectures in Enterprise Workflows

**Tara Bogavelli, Hari Subramani, Roshnee Sharma**
ServiceNow AI Research
**Correspondence:** tara.bogavelli@servicenow.com

## Abstract

While individual components of agentic architectures have been studied in isolation, we lack empirical evidence on how design dimensions interact in realistic multi-agent systems. This study aims to address these gaps by providing a comprehensive enterprise-specific benchmark. We introduce **AgentArch**[1]: a benchmark that evaluates 18 agentic configurations across state-of-the-art large language models. The study spans four dimensions: orchestration strategy, agent style (ReAct versus function calling), memory architecture, and thinking tool integration. Results reveal strong model-specific architectural preferences that challenge one-size-fits-all assumptions in agentic AI. They also expose notable performance gaps on enterprise tasks: the best models reach only 35.3% success on the complex task and 70.8% on the simpler task. We aim to inform architecting choices for enterprise agents by grounding model selection and component design in systematic evidence.

## 1 Introduction

Large language models (LLMs) have evolved beyond simple text generation to function as autonomous agents capable of iterative decision-making and complex task completion (Yao et al., 2025). This transformation has opened new possibilities for enterprise automation, where agentic AI systems can interact with existing workflows, databases, and business processes to accomplish multi-step objectives with minimal human intervention.

Despite growing interest in agentic systems, practitioners face two critical gaps: existing research predominantly evaluates individual components in isolation—orchestration strategies, prompting approaches, memory management, etc—without examining how these architectural choices interact with each other in real-world scenarios. Additionally, while existing benchmarks span diverse domains from gaming to research tasks, few focus specifically on enterprise workflows. Enterprise use cases are uniquely constrained by the need for reliability and seamless integration with existing business systems. The fragmented approach of prior benchmarks leaves enterprise developers without empirical guidance for selecting optimal combinations of architectural components or choosing appropriate models for their specific setups.

This paper presents a comprehensive evaluation of 18 distinct agentic architectures across state-of-the-art LLMs, systematically examining the interactions between four key architectural dimensions: orchestration strategy (single agent vs. multi-agent), agent style (function calling vs. ReAct (Yao et al., 2023)), memory management (complete vs. summarized), and thinking tool integration. We test each architectural combination on two enterprise use cases of varying complexity, providing practical insights for agentic system designers.

Our findings reveal significant model-specific architectural preferences, challenging assumptions about universally optimal designs. We find that even between use cases, models perform best on different architectures. While larger models display more robust performance across architectures, we find that on simpler tasks, smaller models can match performance under their best performing architecture. We also observe significant weaknesses across models when ReAct prompting is used in multi-agent systems. Our results provide empirically-grounded guidance for practitioners building enterprise agentic systems, enabling more informed decisions about architectural components and model selection based on specific use case requirements.

To summarize, our work makes the following contributions:

[1] https://github.com/ServiceNow/AgentArch



*Proceedings of the 15th International Joint Conference on Natural Language Processing and the 5th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pages 915–933
November 6-10, 2026 ©2026 Association for Computational Linguistics

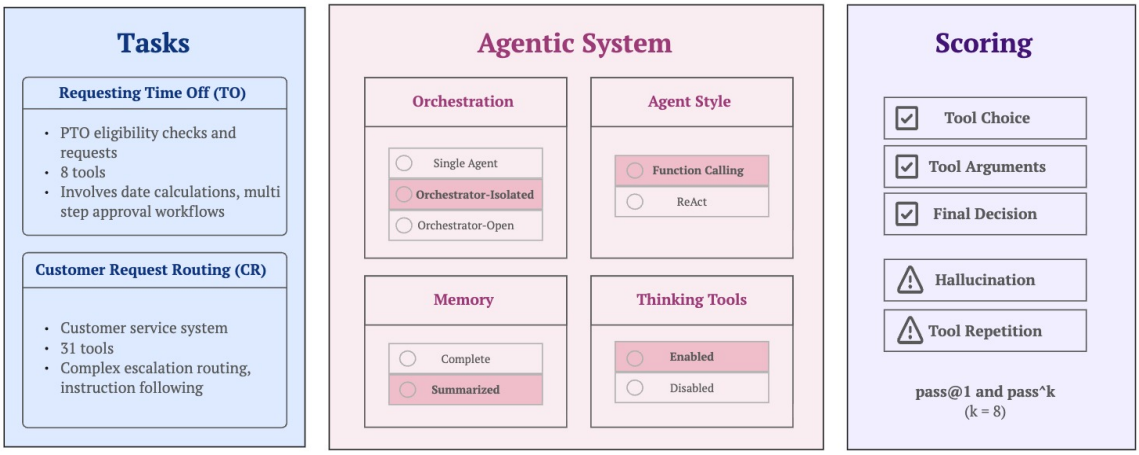



Figure 1: **Overview of AgentArch**. Each of two enterprise use cases (Requesting Time Off and Customer Request Routing) is evaluated across agentic systems that vary along four dimensions: orchestration strategy (single agent, orchestrator-isolated, or orchestrator-open), agent style (function calling or ReAct), memory management (complete or summarized), and thinking tool availability (enabled or disabled). Highlighted options show one example configuration. Each run is scored on tool choice, tool arguments, and final decision, and counts as acceptable only if all three are correct. Hallucination and tool repetition are tracked as additional metrics, and reliability is measured with pass@1 and passˆk, k= 8 trials.

- We introduce **AgentArch**, the first benchmark systematically evaluating 18 agentic architectures across six large language models on realistic enterprise workflows.
- We jointly analyze the effects of orchestration, prompting style, memory design, and thinking tool integration, providing quantitative insight into how these factors interact in practice.
- Our experiments reveal model-specific architectural preferences, showing that optimal configurations vary by model and task complexity, rather than following a single best-performing design.
- We offer evidence-based recommendations for selecting models and architectural configurations suited to enterprise reliability and integration constraints.

Overall, **AgentArch** contributes a systematic, empirically grounded step toward understanding how architectural design choices influence agentic system behavior, supporting more informed and reproducible development practices.

## 2 Related Work

Evaluating agentic AI systems has become a central research focus, yet many studies still assess components in isolation rather than integrated end-to-end performance.

**Tool calling.** Tool use is foundational for agentic systems. Benchmarks such as AgentBench (Liu et al., 2024b), BFCL (Patil et al., 2025), and Seal-Tools (Wu et al., 2024) document persistent weaknesses in tool selection, argument fidelity, output formatting, latency, and redundant calls. More complex settings further degrade performance: combining tool use with intricate instruction following (Yao et al., 2025) or multi-step, nested tool invocations (Basu et al., 2025) increases error rates.

**Memory and context management.** Limited context windows and growing conversational and tool-response histories create pressure on memory design. Prior work explores tiered memory as an OS abstraction (Packer et al., 2024), dynamic knowledge graphs for retrieval and linking (Xu et al., 2025), and dual-track (short/long-term) schemes (Liu et al., 2024a). Empirical studies highlight how tool calls inflate context via frequent, verbose outputs (Maharana et al., 2024), motivating mechanisms that preserve relevance while controlling prompt length.

**Multi-Agent Systems (MAS).** With increasing workflow complexity, MAS introduce specialization and coordination. Benchmarks examine planning structures (hierarchical vs. distributed) and execution quality (Zhu et al., 2025), as well as

| | Requesting Time Off (TO) | Customer Request Routing (CR) |
|---|---|---|
| **Description** | Simple workflow for PTO eligibility and request processing. Tests basic multi-step reasoning and counting. | Complex workflow mimicking customer service systems. Automates simple requests, escalating complex issues. Tests classification and instruction following. |
| **Tools** | 8 custom tools | 31 custom tools |
| **Agents** | Employee Information Retriever, Balance Eligibility, Leave Approval | Request Validation, Duplicate Case Detection, Case Creation, Informational Queries, Transactional Queries, Intent and Sentiment Extraction, Entity Extraction, Document Verification, Email Generation |
| **Key Challenges** | Date calculations; leave balance verification; policy compliance; multi-step approval workflow | Appropriate escalation decisions; context preservation; handling ambiguous requests; complex routing logic; instruction following |

Table 1: Detailed comparison of the two enterprise use cases evaluated in our benchmark: Requesting Time Off (TO) and Customer Request Routing (CR)

inter-agent planning/coordination (Agashe et al., 2025; Geng and Chang, 2025), identifying planning reliability as a bottleneck. Routing frameworks improve agent/model selection and communication patterns (Yue et al., 2025); adaptive approaches search multi-agent configurations online (Hou et al., 2025). Automatic scaling from single agents to MAS can yield gains (Yuan et al., 2025), and heterogeneous-model MAS leverage complementary strengths (Ye et al., 2025).

**Enterprise evaluations.** Enterprise-focused benchmarks consistently reveal a gap between academic evaluations and real deployments. Even state-of-the-art models struggle on routine workplace tasks (Styles et al., 2024; Drouin et al., 2024). CRMArena (Huang et al., 2025a) and CRMArena-Pro (Huang et al., 2025b) evaluate agents on Salesforce-based service, sales, and quote-generation tasks, with CRMArena-Pro showing sharp drops in multi-turn settings, and ITBench (Jha et al., 2025) evaluates agents on IT automation scenarios spanning site reliability, compliance, and financial operations. These benchmarks establish that enterprise tasks remain difficult, but they compare models under a fixed agent design or, in CRMArena, vary only prompting style, finding that function calling helps stronger models but not weaker ones. AgentArch instead holds enterprise workflows fixed and jointly varies orchestration, agent style, memory, and thinking tools.

Prior work has advanced understanding of individual components—tool use (Yao et al., 2025; Patil et al., 2025; Basu et al., 2025), memory (Maharana et al., 2024), and coordination (Geng and Chang, 2025; Agashe et al., 2025)—but remains fragmented. Enterprise-oriented benchmarks capture task complexity yet rarely analyze *interactions* among orchestration, prompting style, memory strategy, and reasoning aids. Our benchmark fills this gap by jointly varying these dimensions across two enterprise use cases, providing evidence on model-specific architectural preferences and the relative advantages of multi-agent versus single agent designs.

## 3 Methodology

### 3.1 Experiment Set Up

**Use Cases** We evaluate two enterprise workflows of differing complexity to examine how task complexity affects agentic performance on real-world enterprise automation scenarios. Both of these use cases involve highly-specific directions to complete tasks in a given order, reflecting the rigid nature of many enterprise workflows where steps must take place in a predefined order. For each request in a use case, we define an expected final decision (e.g. leave approved or rejected). We create mock data so that each tool request returns a deterministic result. Unlike existing benchmarks that use simplified, clean responses, we deliberately construct enterprise-realistic data that is complex, lengthy, and messy. Knowledge base articles span thousands of words with extensive documentation reflecting real enterprise systems. Tool responses return complex JSON objects where relevant information is often buried within metadata, error codes, and auxiliary fields. This approach better represents the information processing challenges agents face in production environments where data is rarely clean or perfectly structured.

**Requesting Time Off (TO)** represents a simple workflow involving PTO eligibility verification and request processing. This use case tests basic multi-

| Orchestration | Agent | Memory | Thinking | GPT-4.1 | | GPT-4o | | GPT-4.1-mini | | o3-mini | | LLaMA 3.3 70B | | Sonnet 4 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | TO | CR | TO | CR | TO | CR | TO | CR | TO | CR | TO | CR |
| **Orch-Open** | React | Complete | Enabled | 36.9 | 21.2 | 6.2 | 0.0 | 7.1 | 0.0 | 1.2 | 0.0 | 0.4 | 0.0 | 28.1 | 0.0 |
| | | Complete | Disabled | 31.9 | 17.5 | 4.8 | 1.2 | 7.3 | 0.5 | 1.2 | 0.0 | 0.0 | 0.0 | 28.7 | 0.2 |
| | FC | Complete | Enabled | 58.8 | 14.8 | 50.2 | 4.2 | 65.8 | 4.8 | 4.6 | 20.5 | 0.0 | 0.0 | 67.7 | 15.2 |
| | | Complete | Disabled | 54.4 | 16.2 | 46.5 | 5.0 | 56.0 | 4.8 | 5.4 | 21.8 | 0.0 | 0.0 | 57.7 | 17.2 |
| | | Summarized | Enabled | 51.9 | 15.2 | 49.0 | 0.8 | 59.0 | 2.5 | 5.0 | 18.5 | 0.0 | 0.0 | 59.6 | 16.8 |
| | | Summarized | Disabled | 52.7 | 15.8 | 45.0 | 0.8 | 49.6 | 1.0 | 4.8 | 19.0 | 0.0 | 0.0 | 44.2 | 16.5 |
| **Orch-Isolated** | React | Complete | Enabled | 28.7 | 17.2 | 3.8 | 0.0 | 4.6 | 0.0 | 2.3 | 0.0 | 0.2 | 0.0 | 28.3 | 0.0 |
| | | Complete | Disabled | 25.8 | 16.0 | 5.8 | 0.2 | 5.6 | 0.0 | 1.5 | 0.0 | 0.0 | 0.0 | 28.1 | 0.5 |
| | FC | Complete | Enabled | 58.8 | 17.2 | 53.5 | 1.8 | 67.1 | 0.8 | 6.0 | 22.2 | 0.0 | 0.0 | 68.5 | 14.8 |
| | | Complete | Disabled | 54.6 | 16.2 | 52.1 | 1.8 | 57.1 | 1.5 | 4.0 | 21.0 | 0.0 | 0.0 | 57.5 | 16.2 |
| | | Summarized | Enabled | 54.6 | 16.5 | 46.0 | 1.8 | 59.6 | 2.0 | 2.9 | 20.5 | 0.0 | 0.0 | 58.1 | 16.8 |
| | | Summarized | Disabled | 51.5 | 15.8 | 44.2 | 3.0 | 54.4 | 1.5 | 4.8 | 21.2 | 0.0 | 0.0 | 44.8 | 14.5 |
| **Single Agent** | React | Complete | Enabled | 34.4 | 1.0 | 15.8 | 0.2 | 51.2 | 1.2 | 8.8 | 1.5 | 6.2 | 0.0 | 63.7 | 3.5 |
| | | Complete | Disabled | 36.7 | 3.8 | 9.6 | 0.0 | 41.5 | 1.2 | 2.9 | 2.2 | 12.2 | 0.0 | 52.7 | 9.0 |
| | FC | Complete | Enabled | 68.1 | 22.2 | 39.0 | 3.0 | 35.4 | 0.2 | 56.7 | 1.5 | 0.0 | 0.0 | 66.7 | 33.8 |
| | | Complete | Disabled | 49.6 | 21.2 | 26.9 | 3.5 | 19.8 | 0.2 | 55.8 | 2.0 | 3.8 | 0.0 | 31.7 | 35.0 |
| | | Summarized | Enabled | 70.8 | 21.5 | 37.9 | 2.5 | 33.5 | 0.5 | 55.0 | 1.5 | 0.6 | 0.0 | 64.4 | 33.2 |
| | | Summarized | Disabled | 48.5 | 19.8 | 28.7 | 2.8 | 24.4 | 0.5 | 55.8 | 1.0 | 0.0 | 0.0 | 31.7 | 35.3 |

Table 2: **Pass@1 Acceptable Rate.** Heatmap cells report the Pass@1 *Acceptable* score (higher/darker is better) for each model–configuration pair on the two enterprise tasks: Time Off (**TO**) and Customer Routing (**CR**). Pass@1 is computed over $k = 8$ attempts per configuration. Overall, function-calling configurations tend to outperform ReAct, thinking tools often help on the simpler TO task, and no single architecture dominates across models or tasks.

| Model | TO | | CR | |
|---|---|---|---|---|
| | Mean | SD | Mean | SD |
| GPT-4.1 | 48.2 | 13.0 | 16.1 | 5.5 |
| GPT-4o | 31.4 | 18.7 | 1.8 | 1.5 |
| GPT-4.1-mini | 38.8 | 22.2 | 1.3 | 1.5 |
| o3-mini | 15.5 | 22.3 | 9.7 | 10.1 |
| LLaMA 70B | 1.3 | 3.1 | 0.0 | 0.0 |
| Sonnet 4 | 49.0 | 15.7 | 15.5 | 12.3 |

Table 3: Model consistency analysis - mean and standard deviation for acceptable pass@1 scores. Performance statistics across all 18 architectural configurations for each use case.

step reasoning, counting, and decision-making with clear success criteria, employing 8 custom tools and 3 agents.

**Customer Request Routing (CR)** represents a complex workflow that mimics intelligent customer service systems, handling simple requests automatically while escalating complex issues to human agents. This tests nuanced classification, context understanding, and appropriate escalation decisions with 31 custom tools and 9 agents. Each use case includes 60 user utterances designed to cover the full spectrum of real-world requests, including successful scenarios, edge cases, and failure conditions that would occur in actual enterprise deployments. Implementation details are provided in Appendix A.

**Orchestration Strategies**. We evaluate three orchestration strategies to examine single agent versus multi-agent performance across key dimensions: decision-making with varying option sets, collaboration costs and benefits, and specialist versus generalist agent effectiveness. **1) Orchestrator-led, Isolated Agent.** An orchestrator assigns tasks to specialist agents and controls all inter-agent communication. When agents need additional information, they request help through the orchestrator, who selects an appropriate agent to respond. This tests whether centralized planning and controlled information flow improve outcomes. **2) Orchestrator-led, Open Agent Network.** The orchestrator assigns initial tasks, but agents communicate directly with each other when additional information is needed. Agents select collaborators independently using agent descriptions. This examines whether direct agent-to-agent communication is more efficient than orchestrator-mediated interaction. **3) Single Agent.** A single agent receives the user request and has access to all available tools, with no collaboration mechanisms. This tests agent performance in larger, unfiltered option spaces.

**Agents**. We evaluate two prevalent agent styles

to examine how different prompting paradigms influence model performance. With **function calling**, the model directly selects tools from the available toolkit, leveraging inherent function calling capabilities without explicit intermediate reasoning. Using **ReAct**, the model outputs its reasoning before selecting actions, following a structured reasoning-action framework (Yao et al., 2023). We investigate whether the explicit reasoning in ReAct prompting yields substantially different outcomes compared to direct function calling. Implementation details and prompts are provided in the appendix.

**Memory Management**. We evaluate two information-sharing approaches to examine the trade-off between context completeness and input context length. With **complete memory**, agents see all previous tool calls, parameters, and responses from other agents, maximizing context but increasing prompt length. With **summarized memory**, agents receive only final summaries from previous agents thus reducing context length.This tests whether comprehensive information sharing enhances performance or condensed summaries suffice for effective collaboration.

**Thinking Tools**. We evaluate whether providing explicit reasoning tools improves model performance on multi-step analysis tasks. When thinking tools are enabled, models receive access to **math** for calculations and **synthesize_collected_information** for data organization, particularly relevant for PTO eligibility, where models must calculate date ranges and compare leave balances. We hypothesize that models without built-in reasoning capabilities will benefit from these structured tools. See Appendix B for more details and examples.

**Model Settings**. We evaluate our benchmark using six models: OpenAI GPT-4.1 (OpenAI, 2025a), OpenAI-GPT-4o (OpenAI, 2024), LLaMA 3.3 70B (AI, 2024), OpenAI GPT-4.1-mini (OpenAI, 2025b), Anthropic Claude Sonnet 4 (Anthropic, 2024), and OpenAI o3-mini (OpenAI, 2025c). We evaluated Claude Sonnet 4 with reasoning disabled. For all experiments, we set the temperature to 0 and use no other sampling parameters. We selected this set of models to represent state-of-the-art models of varying sizes, including models with and without in-built reasoning capabilities. For function calling architectures, we use the in-built OpenAI function calling APIs for all models besides Sonnet 4 which uses the Bedrock API.

### 3.2 Evaluation Metrics

In enterprise agentic systems, reaching the correct conclusion is necessary but insufficient. Consider a leave request scenario: even if the model correctly determines that leave should be approved, the task fails if the model doesn't invoke the approval API, calls it with incorrect parameters (e.g., wrong employee ID), or omits required notifications. Real-world agentic deployment requires that the agentic system **(1)** selects the right tools to complete all required steps, **(2)** invokes them with correct arguments, and **(3)** reaches the correct final outcome.

As such, our primary evaluation metric is the **Acceptable Score**, which measures the percentage of records that satisfy all three critical success conditions simultaneously:

$$\frac{|\{r \in R : C(r) \wedge A(r) \wedge O(r)\}|}{|R|} \times 100\%$$

where $R$ is the set of all test records, and for each record $r$:

- $C(r)$ indicates correct tool choice
- $A(r)$ indicates correct tool arguments ($A(r) = 1.0$)
- $O(r)$ indicates correct final decision

**Tool Choice Evaluation.** We evaluate tool choice under two conditions:

*Lenient Acceptable* ($C_L(r)$): The model executes all required ground truth tools but may perform additional read-only tools. Extraneous write operation tools are penalized.

*Strict Acceptable* ($C_S(r)$): The model must execute exactly the ground truth tools in the correct order with no extraneous tools and zero hallucinations.

**Tool Arguments Scoring.** Tool arguments receive percent scoring where $A(r) = 1.0$ only when all arguments are correct (compared to ground truth).

**Reliability Metrics.** To assess system reliability, we evaluate each configuration over $k = 8$ attempts and compute **pass@1**—the percent of correct samples over $k$ trials. (Guo et al., 2025) and **passˆK**—the probability of *all* $k$ trials succeeding (Yao et al., 2025). The latter metric is particularly important in real-world enterprise domains where reliability and consistency are critical.

| Orchestration / Agent / Memory / Thinking | | | | GPT-4.1 | | GPT-4o | | GPT-4.1-mini | | o3-mini | | LLaMA 3.3 70B | | Sonnet 4 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | TO | CR | TO | CR | TO | CR | TO | CR | TO | CR | TO | CR |
| **Orch-Open** | React | Complete | Enabled | 1.0 | 20.8 | 19.4 | 77.2 | 15.2 | 64.8 | 64.0 | 96.5 | 29.2 | 35.6 | 0.0 | 31.0 |
| | | Complete | Disabled | 2.1 | 18.8 | 22.5 | 71.5 | 18.3 | 65.2 | 67.5 | 98.0 | 28.3 | 30.1 | 0.0 | 26.8 |
| | FC | Complete | Enabled | 0.0 | 0.0 | 42.9 | 16.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | | Complete | Disabled | 0.0 | 0.0 | 58.3 | 18.8 | 0.0 | 0.0 | 0.0 | 0.0 | 0.2 | 0.0 | 0.0 | 0.0 |
| | | Summarized | Enabled | 0.0 | 0.0 | 49.6 | 7.2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | | Summarized | Disabled | 0.0 | 0.0 | 48.1 | 4.8 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **Orch-Isolated** | React | Complete | Enabled | 2.3 | 13.8 | 35.0 | 77.8 | 11.0 | 63.2 | 68.1 | 96.8 | 24.4 | 22.1 | 0.0 | 36.2 |
| | | Complete | Disabled | 4.4 | 19.8 | 31.0 | 86.8 | 16.5 | 62.5 | 66.2 | 97.5 | 23.3 | 28.7 | 0.2 | 31.8 |
| | FC | Complete | Enabled | 0.0 | 0.0 | 0.4 | 19.2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | | Complete | Disabled | 0.0 | 0.0 | 0.2 | 10.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | | Summarized | Enabled | 0.0 | 0.0 | 0.0 | 8.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | | Summarized | Disabled | 0.0 | 0.0 | 0.0 | 4.2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.2 | 0.0 | 0.0 | 0.0 |
| **Single Agent** | React | Complete | Enabled | 0.0 | 0.0 | 0.0 | 11.8 | 0.0 | 0.5 | 0.0 | 0.0 | 0.2 | 0.0 | 0.0 | 0.0 |
| | | Complete | Disabled | 1.9 | 0.2 | 4.2 | 8.5 | 0.2 | 0.0 | 0.0 | 0.0 | 3.3 | 0.0 | 0.0 | 0.0 |
| | FC | Complete | Enabled | 0.0 | 0.0 | 0.0 | 0.2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | | Complete | Disabled | 0.0 | 0.0 | 0.0 | 0.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | | Summarized | Enabled | 0.0 | 0.0 | 0.0 | 0.2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | | Summarized | Disabled | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

Table 4: **Pass@1 Hallucination Rate (↓).** Heatmap cells show the percentage of trials (lower is better, darker indicates higher error) in which an agent hallucinated non-existent entities or schema, e.g., selecting a tool or agent not present in the registry or inventing parameters not defined by a tool schema. Rates are computed at Pass@1 over $k = 8$ attempts per model–configuration on both tasks (TO, CR). Hallucinations tend to concentrate in ReAct settings, especially in multi-agent orchestration.

We use **Acceptable pass@1** as our primary metric. We use lenient correct tool choice for the acceptable score as opposed to strict as strict scoring remains extremely challenging for most models. We use **pass@1** as it brings stability to the lenient acceptable metric over $k$ attempts, representing the success rate for a given model and configuration.

**Additional Metrics** We track supplementary behavioral metrics, including:

- *Hallucination rate*: Frequency of selecting non-existent agents or tools
- *Tool Repetition rate*: Frequency of consecutive identical tool calls with identical arguments

## 4 Results

### 4.1 Overall Results

Complex enterprise workflows remain challenging for current agentic systems. On the simpler time-off task (TO), peak performance reaches 70.8% (GPT-4.1), while the more complex customer routing task (CR) peaks at 35.3% (Sonnet 4). Only GPT-4.1, Sonnet 4, and o3-mini show meaningful capability on CR; most other models score in the single digits. Performance varies markedly across architectures, with no universally best configuration. Models also peak on different configurations across the two tasks, indicating that optimal design depends on task complexity.

With regards to architecture elements, **thinking tools** generally improved performance for non-reasoning models on the simpler time off task. GPT-4.1 improved from 48.5% to 70.8% with thinking tools enabled in single agent function calling with summarized memory. However, thinking tools provided minimal benefit for o3-mini (55.8% to 56.7%) in its best performing category. Thinking tools showed minimal impact on the complex customer routing use case across all models. Compared to other elements, variation in **memory management** styles and **orchestration strategies** (within multi-agent architectures) had minimal impact on scores for most models.

**Function calling** generally outperformed **ReAct** across most models. The notable exception was LLaMA 3.3 70B, which achieved its peak score of 12.2% using single agent ReAct on the requesting time off task. **Multi-agent ReAct** consistently underperformed indicating that ReAct may be better suited towards single agent systems. **Hallucinations** for all models (except for GPT-4o) were

almost exclusively found under ReAct settings suggesting that while models may be robust to hallucinations on the function calling schemas they have been trained on, they still have the propensity to hallucinate in ReAct settings. This is particularly relevant on the more complex use case in multi-agent systems. Sonnet 4, for example, showed hallucination rates of 27-36% on ReAct in multi-agent systems but 0% hallucination in every other configuration. See Table 4 for hallucination rates.

While some models achieved their highest overall scores on single agent architectures, the **correct final decision** scores for the complex customer routing use case were generally higher for multi-agent architectures. Although Sonnet 4's highest overall scores were all achieved with single agent function calling (all scores above 33%), its highest correct final decision scores were for function calling with multi-agent systems (84-87%), while for single agent, it was in the 72-76% range. GPT-4.1 showed similar results as it achieved its peak overall score in both use cases with single agent function calling; however, on the customer routing task, it achieved correct final decision rates of 97-99% with multi-agent function calling and only 79-86% with single agent function calling. GPT-4.1-mini also achieved significantly higher correct final decision rates with multi-agent function calling than with single agent function calling on both use cases. This suggests that while some models are more likely to choose the correct tool or tool arguments on single agent systems, multi-agent systems may be more effective for selecting the correct final decision after all tool calls have been made. See Table 5 for more details.

Two key patterns emerged from the **consistency analysis** shown in Table 3. Sonnet 4 and GPT-4.1 showed the most robust performance with relatively low coefficients of variation (32.1% and 27.0%, respectively, for the simpler task), making them the most reliable choices across different agentic configurations. Conversely, o3-mini exhibited extreme sensitivity to architectural choices (CV = 143.7%).

**Reliability Analysis**. Pass^k scores across all models and agentic configurations peak at 0.0634, indicating only a 6.34% chance of executing the workflow correctly in all 8 trials. In enterprise settings, where reliability and consistency are essential, pass@1 and pass^k results reveal a fundamental gap between the promise of agentic LLMs and their real-world performance. See Appendix E for full scores.

### 4.2 Model-Specific Results

Sonnet 4 achieved the highest scores on the complex CR task with a peak score of 35.3% using single agent function calling. Sonnet 4 performed strongly on the TO task (68.5% using orchestrator-isolated function calling). Notably, thinking tools had a substantial impact on the simpler task but a minimal effect on the CR task. TO scores improved from 31.7% to 64.4% with thinking tools (single agent, summarized), while CR performance remained relatively flat across single agent function calling configurations (33.2%-35.2% range).

GPT-4.1 achieved the highest overall score on the simpler use case - 70.8% using single agent function calling with summarized memory and thinking tools. The model showed sensitivity to thinking tools in single agent configurations, improving from 48.5% to 70.8% when enabled. However, thinking tools provided minimal benefit in multi-agent setups (54.4% vs 58.8%). Memory type preferences varied by orchestration: single agent performance was nearly identical with complete (68.1%) versus summarized (70.8%) memory, while orchestrated configurations slightly favored complete memory.

o3-mini demonstrated extreme architectural sensitivity, achieving 56.7% on time off requests with single agent function calling but dropping to 1.2% with orchestrated ReAct. Thinking tools showed negligible differences (55.8% vs 56.7%). The model achieved its highest CR score of 22.2% using orchestrator-isolated function calling, substantially outperforming single agent performance (1.5%-2.0%). This suggests o3-mini's performance depends heavily on task complexity rather than inherent suitability to single versus multi-agent architecture.

GPT-4o and GPT-4.1-mini peaked with orchestrator-isolated function calling (53.5% and 67.1% TO) using full memory and thinking tools. Both gained from thinking tools on simple tasks—GPT-4o rose from 26.9% to 39.0% in single-agent setups, though gains faded under orchestration. GPT-4o struggled with ReAct, while GPT-4.1-mini scored higher on ReAct (41.5–51.2%) than function calling (19.8–35.4%) in single-agent mode, underscoring how agentic elements interact differently across architectures.

LLaMA 3.3 70B performed poorly overall, scoring 0 on most configurations. Its best result was 12.2% on the TO task with single-agent ReAct (no

| Orchestration | Agent | Memory | Thinking | GPT-4.1 | | GPT-4o | | GPT-4.1-mini | | o3-mini | | LLaMA 3.3 70B | | Sonnet 4 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | TO | CR | TO | CR | TO | CR | TO | CR | TO | CR | TO | CR |
| Orch-Open | React | Complete | Enabled | 71.0 | 75.0 | 14.6 | 30.2 | 10.6 | 4.0 | 15.6 | 5.8 | 1.7 | 0.0 | 87.9 | 68.0 |
| | | Complete | Disabled | 70.8 | 73.2 | 13.3 | 26.2 | 10.6 | 3.8 | 13.8 | 3.8 | 1.0 | 0.0 | 86.5 | 65.2 |
| | FC | Complete | Enabled | 73.3 | 97.5 | 69.0 | 64.0 | 83.3 | 89.5 | 83.3 | 92.2 | 0.0 | 0.0 | 75.2 | 85.2 |
| | | Complete | Disabled | 66.0 | 98.5 | 64.8 | 64.8 | 77.7 | 89.0 | 83.5 | 91.8 | 0.0 | 0.0 | 64.2 | 87.5 |
| | | Summarized | Enabled | 74.8 | 99.2 | 69.8 | 64.8 | 79.8 | 86.8 | 83.8 | 89.0 | 0.0 | 0.0 | 76.0 | 86.8 |
| | | Summarized | Disabled | 69.4 | 98.5 | 68.1 | 64.0 | 74.6 | 84.5 | 81.9 | 91.0 | 0.0 | 0.0 | 67.3 | 86.5 |
| Orch-Isolated | React | Complete | Enabled | 71.9 | 83.5 | 11.5 | 33.0 | 7.3 | 4.5 | 15.8 | 4.2 | 0.4 | 0.0 | 88.3 | 63.7 |
| | | Complete | Disabled | 72.7 | 82.0 | 13.8 | 31.2 | 8.5 | 5.0 | 14.4 | 4.0 | 0.8 | 0.0 | 88.5 | 70.5 |
| | FC | Complete | Enabled | 71.7 | 99.2 | 66.7 | 65.0 | 85.2 | 92.2 | 84.2 | 94.2 | 0.0 | 0.0 | 76.5 | 84.5 |
| | | Complete | Disabled | 72.1 | 97.8 | 68.1 | 64.8 | 79.2 | 92.5 | 82.7 | 91.5 | 0.0 | 0.0 | 64.2 | 85.8 |
| | | Summarized | Enabled | 79.2 | 97.8 | 67.3 | 64.5 | 83.8 | 90.0 | 82.9 | 92.5 | 0.0 | 0.0 | 74.0 | 86.8 |
| | | Summarized | Disabled | 75.4 | 97.8 | 64.4 | 66.5 | 75.0 | 92.8 | 82.9 | 91.0 | 0.0 | 0.0 | 66.9 | 84.0 |
| Single Agent | React | Complete | Enabled | 73.5 | 57.5 | 41.7 | 32.5 | 59.8 | 47.5 | 52.7 | 58.5 | 14.6 | 0.0 | 74.4 | 71.5 |
| | | Complete | Disabled | 76.7 | 58.2 | 26.5 | 29.0 | 50.4 | 49.0 | 38.5 | 47.0 | 21.2 | 0.0 | 72.3 | 70.5 |
| | FC | Complete | Enabled | 72.3 | 86.5 | 64.2 | 66.2 | 59.4 | 23.5 | 83.8 | 18.0 | 0.0 | 0.0 | 76.7 | 75.5 |
| | | Complete | Disabled | 69.4 | 79.2 | 62.1 | 60.5 | 42.9 | 21.5 | 84.8 | 18.0 | 17.9 | 0.0 | 44.8 | 72.5 |
| | | Summarized | Enabled | 74.4 | 85.0 | 61.9 | 65.0 | 59.0 | 23.8 | 82.3 | 14.2 | 8.5 | 0.0 | 76.5 | 76.0 |
| | | Summarized | Disabled | 70.4 | 80.8 | 58.8 | 61.8 | 46.0 | 22.8 | 83.3 | 14.2 | 0.0 | 0.0 | 44.6 | 74.0 |

Table 5: **Pass@1 Correct Final Decision Rate.** Heatmap cells report the probability (higher/ darker is better) that a model–configuration returns the correct ground-truth outcome (*approve/deny*, *route/escalate*, etc.). Computed at Pass@1 over $k = 8$ attempts on both tasks (TO, CR). Together with Table 2, this highlights a trade-off between decision accuracy and end-to-end execution reliability.

thinking tools)—the only model to perform best with ReAct. Scores on customer routing were near zero due to persistent instruction-following issues.

### 4.3 Error Analysis

To understand the findings above, we analyze failed traces from GPT-4.1, Sonnet 4, and o3-mini (1,040 traces in total). We classify each trace with GPT-5.2 as an orchestrator routing error, tool selection error, tool argument error, final decision error, hallucination, or compounding error (an early error that triggers later ones).

**Failure modes are model- and task-specific.** The most common failure category is either tool selection or orchestrator routing, depending on the model and task: GPT-4.1's failures on TO are mostly tool selection (62%), Sonnet 4's on TO are mostly orchestrator routing (78%), and tool selection dominates on CR for Sonnet 4 (67%) and o3-mini (71%). Table 6 classifies every trial by execution pattern, showing that models err in opposite directions. Sonnet 4 over-executes on both tasks (42%), whereas o3-mini under-executes (54% on TO, 62% on CR). GPT-4.1 over-executes on TO (28%) but under-executes on CR (56%), where 31 tools and strict multi-step procedures lead it to skip required steps. The most common extra calls on TO are transactional operations beyond the eligibility check (`update_leave_record`, `notify_employee`, `route_for_approval`).

| Model | Task | Correct | Over | Under | Other |
|---|---|---|---|---|---|
| GPT-4.1 | TO | 39.9 | 28.2 | 15.9 | 16.0 |
| GPT-4.1 | CR | 15.3 | 20.4 | 55.7 | 8.5 |
| Sonnet 4 | TO | 36.8 | 42.0 | 6.1 | 15.0 |
| Sonnet 4 | CR | 16.0 | 42.0 | 26.4 | 15.5 |
| o3-mini | TO | 11.0 | 14.4 | 54.4 | 20.2 |
| o3-mini | CR | 8.4 | 15.1 | 61.6 | 14.9 |

Table 6: Tool execution patterns (% of trials across all configurations). Over: extra tools; Under: required tools missing; Other: wrong order or multiple error types.

**Over-execution is mostly scope confusion.** In 50 sampled over-executing trials from best-configuration runs, 78% were scope confusion: models conflate "check eligibility" with "process the full request," completing the eligibility checks and then continuing into approval. Models also ignore tool outputs. For example, `route_for_approval` returns `"Pending check"`, yet agents then call `update_leave_record` with `status: "approved"`, a dangerous failure mode in enterprise settings.

**Multi-agent failures.** Routing biases mirror execution biases: Sonnet 4 over-routes in 46% of TO and 70% of CR trials, while o3-mini under-routes in 54% of TO trials. Attributing failures to the orchestrator, an agent, or both shows that the pattern depends on task complexity. On TO, failures are often confined to one level (31 of 47 Sonnet 4 failures are orchestrator-only, so downstream agents may have succeeded if routed correctly). On CR, cascading failures dominate (37 of 50 for Sonnet 4, 45 of 50 for o3-mini), as a single misroute among 9 agents and 31 tools triggers errors the system does not recover from. With ReAct, free-text agent names frequently lead to routing to agents that were not required, which together with the hallucination rates in Table 4 helps explain why multi-agent ReAct underperforms. We do not isolate ReAct at the orchestrator from ReAct at the agent level.

**Thinking tools help where the task needs arithmetic.** In TO, 45 of 60 records require date arithmetic, while CR requires none. GPT-4.1 and Sonnet 4 call `math` over 600 times each on TO but at most once on CR, and thinking tools raise their average TO success rate by 6.4 and 14.1 points. o3-mini calls `math` only 4 times and gains 0.7 points, consistent with its built-in reasoning substituting for an external scratchpad. On CR, which requires no arithmetic, thinking tools do not help: Sonnet 4 still averages about 2 thinking-tool calls per record (almost all `synthesize`), yet its success rate changes by about $-1$ point. The drop is small, and we cannot tell whether the extra calls cause it.

## 5 Discussion

We now use the above findings to make concrete recommendations for designing agentic enterprise systems.

**Avoiding Multi-Agent ReAct**. One of the few universal trends that this benchmark shows is that models do not perform their best on Multi-Agent ReAct. In our setup, we used ReAct both for the orchestrator to select agents and for the agents to select tools. It is possible that using ReAct for just one instead of both of these components might improve outcomes, but further work would be needed to test this.

**Multi-Agent Systems for Final Decisions**. While some models performed better overall on single agent systems, we find that multi-agent systems are generally more effective at arriving at the correct final decision. For enterprise use cases where arriving at a final decision is more important than exact tool selection, we recommend the use of multi-agent systems.

**Use Case Considerations**. We observe that models tend to achieve peak performance under different architectures depending on the specific use case, rather than consistently performing best under a single architecture. As such, we caution developers against assuming a single architecture is particularly suited for any model and instead recommend testing against several use cases.

**Using Thinking Tools**. Our results indicate that for non-reasoning models, thinking tools can be a useful way for enabling more careful analysis of information collected over the course of an agentic workflow. This is particularly true in contexts where the model must make a calculated decision based on the information it has collected (as is the case with the requesting time off use case). Thinking tools, however, may also add latency, so we recommend developers consider carefully if their use cases require the extra reasoning abilities that these tools provide.

**Model Selection**. While large models perform the best across architectures, our benchmark shows that on simpler tasks, it is possible to achieve strong performances with smaller models if the appropriate architecture is chosen. For example, while GPT-4.1-mini's scores on the time off use case dip as low as 4.6%, its peak acceptable score of 67.1% is comparable to Sonnet 4's peak score of 68.5%.

## 6 Conclusion

We introduce **AgentArch**, a benchmark for evaluating LLMs on a comprehensive set of agentic architectures on enterprise use cases. We test 18 configurations (single agent vs multi-agent, ReAct vs function calling, integration of thinking tools, and memory management styles) and find that even state-of-the-art LLMs struggle to maintain reliable performance on complex enterprise workflows in any setup. We also observe model-specific architectural preferences across LLMs, suggesting that there is no universally optimal agentic architecture. Furthermore, even the best performing models do not demonstrate reliability over multiple trials showing a fundamental gap between the potential of agentic systems and their effectiveness in practice. We hope these findings provide actionable insights that enterprise developers can use to improve their agentic systems.

## Limitations

Our benchmark has several limitations.

**Use Case Diversity and Size**. We evaluate only two enterprise use cases with 60 samples each, which may not generalize across industries or contexts. One use case represents a simple workflow and the other a complex task, but more varied cases would better reveal how task complexity affects architectural performance.

**Model selection**. Due to cost constraints, our experiments are currently limited to six models total, with only one open-source and one reasoning model. Including more reasoning models would clarify whether o3-mini's results generalize to reasoning models as a class.

**Multimodal Enterprise Integration**. Our current evaluation is limited to text-based tool inputs and outputs, while real enterprise environments frequently involve multimodal data processing, including document analysis, image recognition, and file processing workflows. Future work should expand the benchmark to include tools that accept and return images, PDFs, and other file formats commonly encountered in enterprise systems.

**Conversational Capabilities.** Our benchmark intentionally focuses on autonomous workflows without user interaction as this is an important and understudied application of agentic systems, particularly in enterprise contexts. However, many enterprise applications require dynamic user communication and multi-turn conversations. Future evaluations should incorporate both static settings as well as conversational settings where agents must clarify requirements and adapt to changing user needs throughout task execution.

**Multi-Agent ReAct**. Our benchmark shows poor performance on multi-agent ReAct across models, but does not fully illuminate the reason for this poor performance. Future work focusing specifically on Multi-Agent ReAct and testing ReAct for orchestration and tool calling independently in multi-agent systems would provide more insights on whether ReAct can be effectively utilized in multi-agent systems.

**Sampling Parameters**. All experiments use temperature=0 for reproducibility, which may not reflect production deployments where some stochasticity could be beneficial. The interaction between sampling parameters and architectural choices remains unexplored.

**Success Metrics.** Our "Acceptable Score" requires perfect alignment on all three dimensions (tools, arguments, outcome). This strict evaluation may not capture scenarios where partial success provides business value. Additionally, we do not measure efficiency metrics (latency, cost, token usage) which are critical for enterprise deployment decisions.

## Acknowledgments

We are grateful to Sai Rajeswar Mudumba for his careful reading of the manuscript and helpful suggestions and guidance that improved the final version of this work.

## Ethics Statement

AgentArch is intended for research use as a benchmark for evaluating agentic architectures on enterprise workflows. It is released under the Apache 2.0 license. All employee, customer, and case data in the benchmark is synthetic mock data created for this study and contains no real personal or proprietary information. We do not foresee significant risks from releasing the benchmark. However, results on two use cases should not be taken as sufficient evidence that a model or architecture is ready for production. Our findings show low reliability across repeated trials, and we encourage practitioners to apply additional task-specific testing and human oversight before deploying agents that take actions in real enterprise systems.

## References

Saaket Agashe, Yue Fan, Anthony Reyna, and Xin Eric Wang. 2025. LLM-coordination: Evaluating and analyzing multi-agent coordination abilities in large language models. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 8053–8072, Albuquerque, New Mexico. Association for Computational Linguistics.

Meta AI. 2024. Llama 3.3 70b.

Anthropic. 2024. Claude sonnet 4.

Kinjal Basu, Ibrahim Abdelaziz, Kiran Kate, Mayank Agarwal, Maxwell Crouse, Yara Rizk, Kelsey Bradford, Asim Munawar, Sadhana Kumaravel, Saurabh Goyal, Xin Wang, Luis A. Lastras, and Pavan Kapanipathi. 2025. NESTFUL: A benchmark for evaluating LLMs on nested sequences of API calls. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 33538–33547, Suzhou, China. Association for Computational Linguistics.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. 2024. Workarena: How capable are web agents at solving common knowledge work tasks? In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Longling Geng and Edward Y. Chang. 2025. Realm-bench: A benchmark for evaluating multi-agent systems on real-world, dynamic planning and scheduling tasks. *Preprint*, arXiv:2502.18836.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Zhipeng Hou, Junyi Tang, and Yipeng Wang. 2025. Halo: Hierarchical autonomous logic-oriented orchestration for multi-agent llm systems. *Preprint*, arXiv:2505.13516.

Kung-Hsiang Huang, Akshara Prabhakar, Sidharth Dhawan, Yixin Mao, Huan Wang, Silvio Savarese, Caiming Xiong, Philippe Laban, and Chien-Sheng Wu. 2025a. CRMArena: Understanding the capacity of LLM agents to perform professional CRM tasks in realistic environments. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3830–3850, Albuquerque, New Mexico. Association for Computational Linguistics.

Kung-Hsiang Huang, Akshara Prabhakar, Onkar Thorat, Divyansh Agarwal, Prafulla Kumar Choubey, Yixin Mao, Silvio Savarese, Caiming Xiong, and Chien-Sheng Wu. 2025b. Crmarena-pro: Holistic assessment of llm agents across diverse business scenarios and interactions. *arXiv preprint arXiv:2505.18878*.

Saurabh Jha, Rohan R. Arora, Yuji Watanabe, Takumi Yanagawa, Yinfang Chen, Jackson Clark, Bhavya Bhavya, Mudit Verma, Harshit Kumar, Hirokuni Kitahara, Noah Zheutlin, Saki Takano, Divya Pathak, Felix George, Xinbo Wu, Bekir O Turkkan, Gerard Vanloo, Michael Nidd, Ting Dai, and 21 others. 2025. ITBench: Evaluating AI agents across diverse real-world IT automation tasks. In *Forty-second International Conference on Machine Learning*.

Na Liu, Liangyu Chen, Xiaoyu Tian, Wei Zou, Kaijiang Chen, and Ming Cui. 2024a. From llm to conversational agent: A memory enhanced architecture with fine-tuning of large language models. *Preprint*, arXiv:2401.02777.

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. 2024b. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. 2024. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13851–13870, Bangkok, Thailand. Association for Computational Linguistics.

OpenAI. 2024. Gpt-4o system card.

OpenAI. 2025a. Gpt-4.1.

OpenAI. 2025b. Gpt-4.1-mini.

OpenAI. 2025c. Openai o3-mini.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Memgpt: Towards llms as operating systems. *Preprint*, arXiv:2310.08560.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*.

Olly Styles, Sam Miller, Patricio Cerda-Mardini, Tanaya Guha, Victor Sanchez, and Bertie Vidgen. 2024. Workbench: a benchmark dataset for agents in a realistic workplace setting. In *First Conference on Language Modeling*.

Mengsong Wu, Tong Zhu, Han Han, Chuanyuan Tan, Xiang Zhang, and Wenliang Chen. 2024. Seal-tools: Self-instruct tool learning dataset for agent tuning and detailed benchmark. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 372–384. Springer.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-mem: Agentic memory for LLM agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. 2025. $\tau$-bench: A benchmark for Tool-Agent-User interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Rui Ye, Xiangrui Liu, Qimin Wu, Xianghe Pang, Zhenfei Yin, Lei Bai, and Siheng Chen. 2025. X-mas: Towards building multi-agent systems with heterogeneous llms. *Preprint*, arXiv:2505.16997.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. 2025. EvoAgent: Towards automatic multi-agent generation via evolutionary algorithms. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6192–6217, Albuquerque, New Mexico. Association for Computational Linguistics.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. 2025. MasRouter: Learning to route LLMs for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15549–15572, Vienna, Austria. Association for Computational Linguistics.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru 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.

## A Use Cases and Set Up

When evaluating single agent modes, we simply concatenate all the agent instructions for each defined agent and put them within tags to indicate which step the instructions refer to. In this way, agents in the single agent setup have access to all the same information as in multi-agent setups.

Within each use case, we represent a wide array of potential edge cases. For example, in the Requesting Time Off use case we have user requests that should be rejected and some that should be accepted. Within the rejected category, we include requests that should be rejected for a wide variety of reasons such as invalid leave types, conflicting leaves, and most significantly, lack of leave balance. For lack of balance, we make sure to include complex cases where models must correctly count the number of days across multiple months or take into account leap years and the differing number of days in different months.

For each sample in a use case, the ground truth values for expected tool inputs, expected outcome, and expected tool order were human annotated and verified. We construct the use cases in such a way that a limited set of deterministic ground truths options can be determined for every data sample.

## B Thinking Tools Examples

Below are examples of how thinking tools can be used by the model to complete tasks in agentic frameworks. Thinking tools effectively give the model space to generate extra tokens in the format of calling a tool. There is no actual tool that gets executed when these tools are selected. Whatever the model passes as the tool argument is returned as the result and added to memory so the model can use the information later on in the task.

1. **Synthesizing information**

   **tool:** `synthesize_collected_information`
   **args:**

   ```
   synthesized_information
   (str):
   "I need to determine whether
   or not the employee Sarah
   Smith is eligible for leave.
   I see her leave balance is
   18 days but she has already
   requested 4 days which leaves
   14 days. Her current request
   is for 7 days and I see no
   leave conflicts and the type
   of leave is valid, which
   means she should be eligible
   for this current request."
   ```

   **result (same as input):**

   ```
   "I need to determine whether
   or not the employee Sarah
   Smith is eligible for leave.
   I see her leave balance is
   18 days but she has already
   requested 4 days which leaves
   14 days. Her current request
   is for 7 days and I see
   no leave conflicts and the
   type of leave is valid, which
   means she should be eligible
   for this current request."
   ```

2. **Math**

   **tool:** `math`
   **args:**

   ```
   input (str):
   "Sarah Smith requested leave
   from 'all of February 2024
   to the second of March'. I
   ```

```
need to figure out how many
days that is. Since February
has 29 days in 2024 due to
it being a leap year, that
should be 29 days + 2 days in
March, so 31 days."
```

**result (same as input):**

```
"Sarah Smith requested leave
from 'all of February 2024
to the second of March'. I
need to figure out how many
days that is. Since February
has 29 days in 2024 due to
it being a leap year, that
should be 29 days + 2 days in
March, so 31 days."
```

## C AI Usage

We use AI in this paper for generating the code required for running the benchmark, for proof reading and paraphrasing, and for creating the graphics and figure formatting.

## D Prompts

Prompts were designed to be simple and representative examples of both traditional function calling-style prompts and ReAct prompts. For function calling prompts, the tools and directions are placed in the system prompt while the user prompt includes the main objective and conversation history thus far. We also include a note to always call a tool (since communication with the user is not supported), always use real input values (since some models are prone to using placeholders instead), and to select the finish tool when finished since this is critical for control flow as well as getting a self reported summary of each agents actions which is passed to other agents in the summarized memory management style. We attempt to convey the same information in the ReAct style prompt however as is typical with ReAct prompting, all information is in the system prompt and a JSON response is required. The ReAct prompts include a sample JSON output format that asks the model for "thought" (reasoning) and an "action" and "observation.".

## E Additional Metrics

We include supplemental metrics here all shown with pass@1: Correct Tools Strict Rate, Correct Tools Lenient Rate, Tool Repetition Rate (higher scores indicate more repetition), Missing Required Tool Rate (higher scores indicate more missing tools). We also show Passˆ K for the Acceptable Rate.

| Orchestration | Agent | Memory | Thinking Tools | GPT-4.1 | | GPT-4o | | GPT-4.1-mini | | o3-mini | | LLaMA 3.3 70B | | Sonnet 4 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | TO | CR | TO | CR | TO | CR | TO | CR | TO | CR | TO | CR |
| Orch-Open | React | Complete | Enabled | 8.1 | 19.5 | 14.4 | 1.2 | 13.3 | 2.5 | 5.8 | 0.0 | 4.2 | 0.0 | 14.0 | 0.0 |
| | | Complete | Disabled | 5.2 | 15.8 | 11.5 | 1.2 | 10.6 | 3.0 | 6.2 | 0.0 | 2.3 | 0.0 | 21.2 | 0.5 |
| | Function Calling | Complete | Enabled | 52.3 | 17.5 | 27.9 | 18.0 | 14.2 | 16.2 | 2.3 | 18.2 | 0.0 | 0.0 | 58.8 | 16.0 |
| | | Complete | Disabled | 49.6 | **21.5** | 17.1 | **19.8** | 12.1 | 16.8 | 2.1 | 19.8 | 0.0 | 0.0 | 49.8 | 17.2 |
| | | Summarized | Enabled | 49.0 | 17.5 | 24.2 | 17.0 | 15.4 | 19.0 | 2.9 | 16.8 | 0.0 | 0.0 | 61.3 | 16.8 |
| | | Summarized | Disabled | 49.6 | 17.5 | 24.0 | 16.8 | 6.2 | **21.5** | 2.7 | 17.0 | 0.0 | 0.0 | 57.1 | 16.5 |
| Orch-Isolated | React | Complete | Enabled | 10.2 | 16.0 | 8.3 | 0.2 | 12.1 | 3.2 | 7.3 | 0.0 | **6.0** | 0.0 | 12.1 | 0.0 |
| | | Complete | Disabled | 9.8 | 15.2 | 9.8 | 0.0 | 10.4 | 1.2 | 5.0 | 0.0 | 3.3 | 0.0 | 18.8 | 0.0 |
| | Function Calling | Complete | Enabled | 50.0 | 17.5 | 47.1 | 13.8 | 6.0 | 0.8 | 2.5 | **20.2** | 0.0 | 0.0 | 61.0 | 15.0 |
| | | Complete | Disabled | 50.2 | 17.0 | **50.8** | 12.2 | 6.5 | 1.5 | 1.5 | 18.5 | 0.0 | 0.0 | 55.0 | 16.2 |
| | | Summarized | Enabled | 44.4 | 16.5 | 39.0 | 12.8 | **16.9** | 2.0 | 0.8 | 18.5 | 0.0 | 0.0 | **62.3** | 16.8 |
| | | Summarized | Disabled | 44.4 | 16.0 | 43.8 | 14.2 | 10.8 | 1.5 | 2.3 | 19.5 | 0.0 | 0.0 | 57.9 | 15.0 |
| Single Agent | React | Complete | Enabled | 30.0 | 5.2 | 20.8 | 1.8 | 0.2 | 7.2 | 8.8 | 1.5 | 0.8 | 0.0 | 58.8 | 17.2 |
| | | Complete | Disabled | 39.4 | 9.8 | 13.1 | 1.0 | 0.0 | 5.2 | 4.0 | 2.2 | 0.8 | 0.0 | 47.9 | 31.0 |
| | Function Calling | Complete | Enabled | 68.5 | 12.2 | 31.9 | 4.2 | 0.0 | 0.2 | 38.5 | 1.2 | 0.0 | 0.0 | 3.3 | **37.2** |
| | | Complete | Disabled | 52.3 | 18.5 | 26.7 | 4.2 | 0.2 | 0.2 | 37.7 | 2.5 | 0.0 | 0.0 | 0.0 | 36.0 |
| | | Summarized | Enabled | **71.7** | 13.2 | 30.2 | 3.2 | 0.0 | 0.8 | 36.5 | 1.8 | 0.0 | 0.0 | 3.1 | 35.8 |
| | | Summarized | Disabled | 49.0 | 17.2 | 23.5 | 3.2 | 0.0 | 0.2 | **39.2** | 1.2 | 0.0 | 0.0 | 0.0 | 36.2 |

Table 7: **Pass@1 Correct Tools Strict Rate**. Bolded scores indicate highest score per model on use case

| Orchestration | Agent | Memory | Thinking Tools | GPT-4.1 | | GPT-4o | | GPT-4.1-mini | | o3-mini | | LLaMA 3.3 70B | | Sonnet 4 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | TO | CR | TO | CR | TO | CR | TO | CR | TO | CR | TO | CR |
| Orch-Open | React | Complete | Enabled | 37.9 | 21.8 | 26.5 | 7.0 | 32.3 | 19.5 | 8.8 | 0.2 | 16.2 | 4.5 | 28.1 | 30.8 |
| | | Complete | Disabled | 32.1 | 19.0 | 23.1 | 10.2 | 24.4 | 26.0 | 9.2 | 0.0 | 12.7 | 6.6 | 29.0 | 43.2 |
| | Function Calling | Complete | Enabled | 58.8 | 25.8 | 51.2 | 24.2 | 69.2 | 43.8 | 4.6 | 20.5 | 13.3 | 0.0 | 68.3 | 44.0 |
| | | Complete | Disabled | 54.8 | 28.5 | 46.5 | **26.8** | 56.9 | **51.7** | 5.4 | 21.8 | 14.2 | 0.0 | 57.7 | 46.0 |
| | | Summarized | Enabled | 55.6 | 23.2 | **55.4** | 25.5 | 68.3 | 42.2 | 5.2 | 18.5 | 0.0 | 0.0 | 69.8 | 42.0 |
| | | Summarized | Disabled | 55.2 | 25.2 | 51.2 | 24.2 | 54.8 | 46.8 | 5.2 | 19.2 | 0.0 | 0.0 | 57.9 | 43.2 |
| Orch-Isolated | React | Complete | Enabled | 29.2 | 17.8 | 25.0 | 1.0 | 27.1 | 8.8 | 9.4 | 0.5 | **18.1** | 2.0 | 28.5 | 3.2 |
| | | Complete | Disabled | 26.2 | 17.0 | 24.2 | 1.8 | 26.9 | 7.5 | 8.3 | 0.2 | 13.5 | 0.8 | 28.1 | 2.2 |
| | Function Calling | Complete | Enabled | 58.8 | 17.8 | 53.5 | 14.0 | **70.4** | 0.8 | 6.0 | **22.2** | 3.1 | 0.0 | **70.2** | 42.8 |
| | | Complete | Disabled | 55.0 | 17.0 | 52.3 | 13.0 | 58.5 | 1.8 | 4.0 | 21.0 | 0.6 | 0.0 | 57.5 | 16.2 |
| | | Summarized | Enabled | 55.8 | 16.5 | 50.6 | 14.0 | 69.0 | 2.0 | 2.9 | 20.5 | 0.0 | 0.0 | 66.9 | 16.8 |
| | | Summarized | Disabled | 53.8 | 16.0 | 49.0 | 15.2 | 60.0 | 1.5 | 5.0 | 21.2 | 0.0 | 0.0 | 57.9 | 15.0 |
| Single Agent | React | Complete | Enabled | 36.5 | 8.0 | 25.4 | 2.5 | 57.9 | 19.2 | 9.2 | 1.8 | 11.2 | 4.8 | 63.7 | 30.0 |
| | | Complete | Disabled | 46.0 | 16.0 | 19.8 | 3.5 | 59.4 | 13.0 | 4.2 | 2.5 | 12.7 | **7.9** | 54.2 | 35.8 |
| | Function Calling | Complete | Enabled | 69.0 | 29.0 | 40.6 | 5.5 | 43.5 | 12.2 | **56.9** | 1.5 | 0.0 | 0.0 | 66.7 | 43.5 |
| | | Complete | Disabled | 55.2 | **36.5** | 29.4 | 7.2 | 37.5 | 12.8 | 56.0 | 3.0 | 5.0 | 0.0 | 31.7 | **50.7** |
| | | Summarized | Enabled | **71.9** | 29.5 | 40.2 | 6.5 | 41.9 | 13.0 | 55.0 | 2.8 | 0.8 | 0.0 | 64.4 | 42.0 |
| | | Summarized | Disabled | 53.3 | 34.5 | 30.8 | 6.0 | 41.5 | 16.0 | 56.0 | 1.2 | 0.0 | 0.0 | 31.7 | 49.0 |

Table 8: **Pass@1 Correct Tools Lenient Rate**. Bolded scores indicate highest score per model on use case

| Orchestration | Agent | Memory | Thinking Tools | GPT-4.1 | | GPT-4o | | GPT-4.1-mini | | o3-mini | | LLaMA 3.3 70B | | Sonnet 4 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | TO | CR | TO | CR | TO | CR | TO | CR | TO | CR | TO | CR |
| Orch-Open | React | Complete | Enabled | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | | Complete | Disabled | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | Function Calling | Complete | Enabled | 1.4 | 0.0 | 0.4 | 0.0 | 3.5 | **0.0** | 0.0 | 0.0 | 0.0 | 0.0 | 4.4 | 0.0 |
| | | Complete | Disabled | 0.8 | 0.0 | 0.2 | **0.0** | 1.0 | **0.0** | 0.0 | 0.0 | 0.0 | 0.0 | 1.2 | 0.0 |
| | | Summarized | Enabled | 0.5 | 0.0 | 0.3 | 0.0 | 1.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.6 | 0.0 |
| | | Summarized | Disabled | 0.6 | 0.0 | 0.2 | 0.0 | 0.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.0 |
| Orch-Isolated | React | Complete | Enabled | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | | Complete | Disabled | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | Function Calling | Complete | Enabled | 1.4 | 0.0 | **0.7** | 0.0 | **4.1** | 0.0 | 0.0 | **0.0** | 0.0 | 0.0 | **4.9** | 0.0 |
| | | Complete | Disabled | 0.8 | 0.0 | 0.5 | 0.0 | 1.1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.2 | 0.0 |
| | | Summarized | Enabled | 0.8 | 0.0 | 0.2 | 0.0 | 1.6 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.3 | 0.0 |
| | | Summarized | Disabled | 0.5 | 0.0 | 0.1 | 0.0 | 0.8 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.2 | 0.0 |
| Single Agent | React | Complete | Enabled | 0.0 | 0.0 | 0.0 | 0.0 | 0.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.7 | 0.0 |
| | | Complete | Disabled | 0.0 | 0.0 | 0.0 | 0.0 | 0.1 | 0.0 | 0.0 | 0.0 | **0.0** | 0.0 | 0.6 | 0.0 |
| | Function Calling | Complete | Enabled | 4.6 | **0.0** | 0.1 | 0.0 | 0.0 | 0.0 | **1.1** | 0.0 | 0.0 | 0.0 | 3.9 | 0.0 |
| | | Complete | Disabled | 0.4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.9 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | | Summarized | Enabled | **6.3** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.8 | 0.0 | 0.0 | 0.0 | 2.9 | 0.0 |
| | | Summarized | Disabled | 0.3 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.9 | 0.0 | 0.0 | 0.0 | 0.0 | **0.0** |

Table 9: **Pass^ K Acceptable Rate**. Bolded scores indicate highest score per model on use case

| Orchestration | Agent | Memory | Thinking Tools | GPT-4.1 | | GPT-4o | | GPT-4.1-mini | | o3-mini | | LLaMA 3.3 70B | | Sonnet 4 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | TO | CR | TO | CR | TO | CR | TO | CR | TO | CR | TO | CR |
| Orch-Open | React | Complete | Enabled | 7.5 | 28.2 | 57.7 | 36.2 | 55.2 | 93.5 | **64.2** | 1.5 | 90.0 | 84.2 | 0.6 | **56.8** |
| | | Complete | Disabled | 9.4 | **35.3** | **58.1** | **41.8** | 53.8 | **94.0** | **64.2** | **3.2** | 94.6 | 96.8 | 0.0 | 40.2 |
| | Function Calling | Complete | Enabled | 9.4 | 0.0 | 9.4 | 5.0 | 9.6 | 17.5 | 0.0 | 0.0 | **100.0** | **100.0** | 0.0 | 0.0 |
| | | Complete | Disabled | 11.0 | 0.0 | 10.6 | 3.8 | 15.6 | 16.8 | 0.0 | 0.0 | **100.0** | **100.0** | 0.0 | 0.0 |
| | | Summarized | Enabled | 12.5 | 0.0 | 3.8 | 2.8 | 24.0 | 19.5 | 0.0 | 0.0 | **100.0** | **100.0** | 0.0 | 0.0 |
| | | Summarized | Disabled | 13.1 | 0.0 | 4.2 | 1.8 | 31.0 | 18.8 | 0.0 | 0.0 | **100.0** | **100.0** | 0.0 | 0.2 |
| Orch-Isolated | React | Complete | Enabled | 10.6 | 17.2 | 54.0 | 20.0 | 54.6 | 90.0 | 62.1 | 1.0 | 91.5 | 92.4 | **1.5** | 26.0 |
| | | Complete | Disabled | 10.2 | 22.8 | 46.2 | 28.7 | **61.3** | 81.8 | 62.7 | 2.0 | 95.6 | 91.2 | 0.4 | 24.2 |
| | Function Calling | Complete | Enabled | 6.9 | 0.2 | 1.7 | 7.8 | 9.8 | 18.2 | 0.0 | 0.0 | **100.0** | **100.0** | 0.0 | 0.0 |
| | | Complete | Disabled | 12.2 | 0.0 | 1.5 | 2.8 | 21.2 | 13.2 | 0.0 | 0.0 | **100.0** | **100.0** | 0.0 | 0.0 |
| | | Summarized | Enabled | 9.0 | 0.2 | 4.4 | 5.5 | 24.8 | 17.0 | 0.0 | 0.0 | **100.0** | **100.0** | 0.0 | 0.0 |
| | | Summarized | Disabled | **16.0** | 0.0 | 2.3 | 6.5 | 32.5 | 13.0 | 0.0 | 0.0 | **100.0** | **100.0** | 0.0 | 0.0 |
| Single Agent | React | Complete | Enabled | 1.0 | 22.8 | 4.4 | 3.2 | 20.4 | 46.5 | 0.0 | 0.0 | 0.0 | 24.2 | 0.0 | 8.8 |
| | | Complete | Disabled | 1.9 | 21.2 | 2.9 | 5.5 | 9.4 | 29.8 | 0.0 | 0.0 | 0.4 | 12.7 | 0.0 | 8.0 |
| | Function Calling | Complete | Enabled | 0.0 | 5.8 | 0.2 | 6.0 | 0.2 | 4.5 | 0.0 | 0.0 | 0.0 | 88.9 | 0.0 | 0.0 |
| | | Complete | Disabled | 0.4 | 2.8 | 0.0 | 5.0 | 1.5 | 6.5 | 0.0 | **100.0** | 98.8 | **100.0** | 0.0 | 0.0 |
| | | Summarized | Enabled | 0.0 | 4.2 | 0.6 | 5.2 | 0.6 | 3.5 | 0.0 | **100.0** | 99.2 | **100.0** | 0.0 | 0.0 |
| | | Summarized | Disabled | 0.4 | 3.2 | 0.0 | 4.8 | 1.7 | 5.0 | 0.0 | 0.0 | 0.0 | 98.8 | 0.0 | 0.0 |

Table 10: **Pass@1 Tool Repetition Rate ↓.** Bolded scores indicate highest score (most repetition) per model on use case

| Orchestration | Agent | Memory | Thinking Tools | GPT-4.1 | | GPT-4o | | GPT-4.1-mini | | o3-mini | | LLaMA 3.3 70B | | Sonnet 4 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | TO | CR | TO | CR | TO | CR | TO | CR | TO | CR | TO | CR |
| Orch-Open | React | Complete | Enabled | 36.0 | 65.5 | 32.3 | 62.0 | 32.3 | 58.8 | 74.6 | 63.2 | 31.0 | 68.2 | 3.1 | 15.5 |
| | | Complete | Disabled | 43.5 | 65.8 | 34.6 | 60.8 | 36.9 | 50.5 | 70.4 | 62.3 | 41.9 | 85.4 | 3.3 | 5.8 |
| | Function Calling | Complete | Enabled | 13.8 | 58.5 | 5.0 | 56.2 | 6.2 | 27.3 | 94.6 | 74.2 | 85.8 | **100.0** | 0.6 | 40.0 |
| | | Complete | Disabled | 9.0 | 56.5 | 2.7 | 54.2 | 5.0 | 16.2 | 94.0 | 73.8 | 83.5 | **100.0** | 3.8 | 40.0 |
| | | Summarized | Enabled | 7.1 | 61.3 | 4.2 | 54.8 | 5.4 | 27.8 | 94.2 | 74.2 | **100.0** | **100.0** | 2.1 | 40.0 |
| | | Summarized | Disabled | 6.5 | 60.0 | 3.3 | 55.2 | 6.9 | 18.2 | 94.0 | 73.5 | **100.0** | **100.0** | 1.9 | 40.0 |
| Orch-Isolated | React | Complete | Enabled | 55.2 | **70.0** | 40.0 | 72.8 | **43.8** | 70.2 | 76.2 | 63.7 | 28.5 | 69.2 | 2.3 | 66.8 |
| | | Complete | Disabled | **57.5** | 69.8 | 42.5 | 72.8 | 39.0 | **70.5** | 73.3 | 62.3 | 42.1 | 77.8 | 2.7 | 68.2 |
| | Function Calling | Complete | Enabled | 13.5 | 69.8 | 10.8 | 69.8 | 5.0 | 70.0 | 92.5 | 73.0 | 96.9 | 96.4 | 0.6 | 40.2 |
| | | Complete | Disabled | 16.7 | **70.0** | 1.9 | 69.2 | 4.4 | 70.0 | 95.0 | 72.8 | 99.4 | 98.2 | 2.5 | **70.0** |
| | | Summarized | Enabled | 19.4 | **70.0** | 11.2 | 68.8 | 4.0 | 70.0 | **95.6** | 73.0 | **100.0** | 98.8 | 3.1 | **70.0** |
| | | Summarized | Disabled | 17.3 | **70.0** | 5.0 | 69.0 | 2.9 | 70.0 | 94.2 | 74.0 | **100.0** | **100.0** | 3.5 | **70.0** |
| Single Agent | React | Complete | Enabled | 34.0 | 62.0 | 40.8 | 74.0 | 23.5 | 57.0 | 44.6 | 83.5 | 47.1 | 79.2 | 10.6 | 33.8 |
| | | Complete | Disabled | 32.9 | 55.2 | **46.7** | **75.8** | 28.3 | 62.0 | 34.8 | 77.8 | 51.9 | 82.4 | **16.7** | 28.5 |
| | Function Calling | Complete | Enabled | 3.5 | 41.0 | 18.8 | 65.0 | 16.9 | 57.8 | 24.8 | **94.8** | **100.0** | **100.0** | 10.2 | 22.5 |
| | | Complete | Disabled | 16.5 | 33.5 | 24.6 | 62.7 | 18.8 | 57.8 | 26.2 | 92.5 | 91.9 | **100.0** | 11.5 | 19.5 |
| | | Summarized | Enabled | 3.1 | 40.5 | 18.3 | 65.0 | 16.5 | 56.5 | 25.4 | 93.5 | 98.3 | **100.0** | 12.5 | 24.2 |
| | | Summarized | Disabled | 17.3 | 35.5 | 18.3 | 64.8 | 15.4 | 53.5 | 24.8 | 94.2 | **100.0** | **100.0** | 11.7 | 20.2 |

Table 11: **Pass@1 Missing Required Tool Rate ↓**. Bolded scores indicate highest score (most missing tools) per model on use case

**system**:

You are an agent. Your goal is to accomplish your main objective using the directions provided. You must follow the directions you are given. If the directions instruct you to complete steps in a specific order, you must follow that order.

Directions: {agent_instructions}

**user**:

Main Objective: {main_objective}

Conversation History: {memory}

Always call a tool. You must always pass real values as tool inputs. You may never use placeholders. Before calling a tool, remember to consider the conversation history to understand what has already been accomplished and what needs to be done next. When you have finished your task, you MUST select the finish tool. Otherwise your response will be ignored.

Figure 2: Function Calling Agent Prompt

**user**: You are an agent. Your goal is to accomplish your main objective using the directions provided. You must follow the directions you are given. If the directions instruct you to complete steps in a specific order, you must follow that order.

Directions:
{agent_instructions}

Main Objective:
{main_objective}

Conversation History:
{memory}

To accomplish your main objective, you must select an action from the following options:
{tools}

Always select an action using the format below. You must always pass real values as arguments. You may never use placeholders. Before choosing an action, remember to consider the conversation history to understand what has already been accomplished and what needs to be done next. When you have finished your task, you MUST select the finish action. Otherwise your response will be ignored.

Output your response in the following JSON format:

```
[
{
"thought": "<reason step by step about the current situation and what to do next>",
"action": {
"name": "<tool_name_1>",
"arguments": {
"<argument_name>": <argument_value>
}
}
"observation": "<expected result of the tool execution>"
}...
{
"thought": "<reason step by step about the current situation and what to do next>",
"action": {
"name": "<tool_name_m>",
"arguments": {
"<argument_name>": <argument_value>
}
}
"observation": "<expected result of the tool execution>"
}
]
```

Figure 3: ReAct Agent Prompt

**user**: You are the orchestrator of a group of agents. Your job is to select the next agent(s) best suited for accomplishing the request given the conversation history thus far. You must follow the directions you are given. If the directions instruct you to complete steps in a specific order, you must follow that order.
The conversation history below shows you what agents have already done to accomplish the task.

Directions:
{usecase_directions}

Request:
{request}

Conversation History:
{memory}

To accomplish your request, you must select from the following agents:
{agents}

Always select an agent using the format below. You must always pass real values as arguments. You may never use placeholders. Before selecting the next agent(s), consider your request, directions, and the conversation history thus far to understand what to do next.

Output your response in the following JSON format:

```
[
{
"thought": "<reason step by step about the current situation and what to do next>",
"action": {
"name": "<agent_name_1>",
"arguments": {
"main_objective": <main objective you want the agent to accomplish>
}
}
"observation": "<expected result of the agent execution>"
}...
{
"thought": "<reason step by step about the current situation and what to do next>",
"action": {
"name": "<agent_name_n>",
"arguments": {
"main_objective": <main objective you want the agent to accomplish>
}
}
"observation": "<expected result of the agent execution>"
}
]
```

If there are no relevant agents, respond with "Unfortunately no agents are able to help with this request." Otherwise you MUST respond with at least one agent in the format above. .

Figure 4: Orchestrator ReAct Prompt

**system**:
You are the orchestrator of a group of agents. Your job is to select the next agent(s) best suited for accomplishing the request. You must follow the directions you are given. If the directions instruct you to complete steps in a specific order, you must follow that order. When you select an agent, you must give it a main objective that describes what the agent should accomplish.

You also have access to the conversation history below which shows you what agents have already done to accomplish the task.

Directions: {usecase_directions}

**user**:
Request: {user_utterance}

Conversation History: {memory}

Always select an agent. You must always pass real values as arguments. You may never use placeholders. Before selecting the next agent(s), consider your request, directions, and the conversation history thus far to understand what to do next. If there are no relevant agents, respond with "Unfortunately no agents are able to help with this request." Otherwise you MUST respond with at least one agent .

Figure 5: Orchestrator Function Calling Prompt