MCP-AgentBench: Evaluating Real-World Language Agent Performance with MCP-Mediated Tools

Zikang Guo[1*], **Benfeng Xu**[1,2], **Chiwei Zhu**[1], **Wentao Hong**[2], **Xiaorui Wang**[2], **Zhendong Mao**[1†]
[1]University of Science and Technology of China
[2]MetastoneTechnology, Beijing, China
{gzk170401, benfeng}@mail.ustc.edu.cn

The Model Context Protocol (MCP) is rapidly emerging as a pivotal open standard, designed to enhance agent-tool integration and interoperability, and is positioned to unlock a new era of powerful, interconnected, and genuinely utilitarian agentic AI. However, despite MCP's growing adoption, existing benchmarks often fail to capture real-world agent performance within this new paradigm, leading to a distorted perception of their true operational value and an inability to reliably differentiate proficiencies. To bridge this critical evaluation gap, we introduce MCP-AgentBench—a comprehensive benchmark specifically engineered to rigorously assess language agent capabilities in MCP-mediated tool interactions. Core contributions of MCP-AgentBench include: the establishment of a robust MCP testbed comprising 33 operational servers with 188 distinct tools; the development of a benchmark featuring 600 systematically designed queries distributed across 6 distinct categories of varying interaction complexity; and the introduction of MCP-Eval, a novel outcome-oriented evaluation methodology prioritizing real-world task success. Through extensive empirical evaluation of leading language agents, we provide foundational insights. MCP-AgentBench aims to equip the research community with a standardized and reliable framework to build, validate, and advance agents capable of fully leveraging MCP's transformative benefits, thereby accelerating progress toward truly capable and interoperable AI systems.

1 Introduction

Language agents, leveraging Large Language Models (LLMs) for reasoning and interaction [27, 30, 10, 26], are rapidly emerging as a transformative force in AI. Their ability to autonomously operate in digital environments, including navigating the web [40, 23, 33], controlling applications [32, 19, 2], and interfacing with software tools [35, 16], signals significant advancements. However, a primary hurdle to widespread agent utility is achieving effective and scalable interaction with the external world. This involves not only agent-level mastery of complex tool-use workflows [24, 25, 35] (from intent understanding to multi-step orchestration) but also resolving the ecosystem-level "M×N integration problem", where M agents must interface with N disparate tools, creating a complex and fragmented landscape that hinders interoperability for cohesive, large-scale agentic systems.

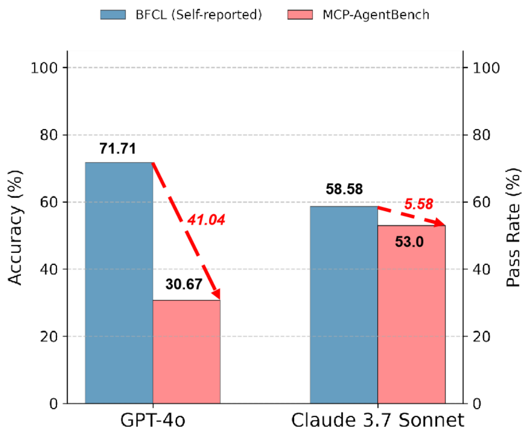


Figure 1: BFCL [34] vs. MCP-AgentBench

To address these obstacles, the Model Context Protocol (MCP) [4] has emerged as a pivotal open standard for AI-tool interactions, providing a universal communication layer and standardized interface to simplify the integration landscape. Crucially, MCP distinguishes itself from traditional function calling [25, 36] approaches in several key aspects. Firstly, while function calling typically treats tools as discrete, callable units,

[*]Work done during the internship at Metastone.
[†]Corresponding author: Zhendong Mao.

MCP is designed to facilitate interaction with a server as a more holistic entity, potentially managing context or state across multiple operations based on the protocol's specifications. Secondly, MCP standardizes the feedback mechanism from the environment, ensuring agents receive richer, more actionable, and consistent responses that go beyond simple function return values, which is vital for robust autonomous operation. Thirdly, unlike paradigms requiring developers to meticulously pre-define tool schemas for the agent, MCP's standardized nature is envisioned to support more dynamic tool discovery and interaction, with servers exposing their capabilities in a universally understandable format, thus reducing bespoke integration efforts and enhancing agent adaptability. By fostering such an open, interoperable, and scalable environment, MCP lays the foundation for a new generation of autonomous and deeply integrated AI experiences, positioning it as a fundamental enabler for powerful, interconnected, and utilitarian agentic AI [12].

However, MCP's emergence necessitates an evolution in agent evaluation. Existing function calling benchmarks [34, 25, 36, 39], lacking consideration for such standardized interaction protocols, are ill-suited for MCP-mediated activities. Consequently, their failure to accurately measure effectiveness or reliably differentiate capabilities creates a critical evaluation gap, hindering both the development of MCP-based systems and systematic progress in leveraging MCP's transformative benefits. For instance, scores from benchmarks like the BFCL [34], which typically evaluate a narrower scope of tool interactions (Figure 1), can misrepresent practical efficacy. While real-world user experiences (e.g., on OpenRouter [22]) suggest models such as Claude 3.7 Sonnet [3] can outperform GPT-4o [20] in complex tool-assisted tasks, their BFCL rankings may significantly diverge. Such discrepancies underscore the limitations of current benchmarks in gauging true agent effectiveness, particularly for standardized, protocol-driven interactions as promoted by MCP, thereby risking misinterpretations of model capabilities.

To bridge this critical evaluation gap, we introduce MCP-AgentBench, a comprehensive benchmark engineered to assess language agent proficiency in MCP-mediated tool use. Our benchmark relies on a meticulously developed MCP server testbed, a unified platform with diverse, deployed MCP-compliant servers, providing a realistic and standardized evaluation environment. This testbed comprises 33 MCP servers offering 188 tools, selected based on stringent criteria of executability, statelessness (to ensure reproducibility in initial evaluations), and text-based interaction.

MCP-AgentBench features systematically categorized queries spanning a spectrum of interaction complexities, from single-server operations to multi-server sequential workflows requiring sophisticated planning and information synthesis. Its data construction follows a rigorous three-stage process: (1) establishing the MCP server testbed; (2) generating diverse, realistic queries via a structured categorization framework that systematically varies interaction complexity (e.g., number of servers, inter-server dependencies, reasoning depth); and (3) creating reference answers using an LLM-assisted, human-verified annotation process.

To evaluate agent performance objectively, we developed MCP-Eval, an LLM-as-a-judge [38] methodology tailored for MCP-AgentBench. This approach prioritizes real-world effectiveness by assessing tangible task success rather than rigid adherence to specific execution paths, recognizing that multiple valid solution trajectories often exist for complex problems.

In summary, our main contributions are:

1. The establishment of a comprehensive MCP server testbed, integrating diverse MCP-compliant servers under a unified system to provide a standardized environment for MCP research and agent development.

2. MCP-AgentBench, a comprehensive benchmark built on this testbed, designed to rigorously evaluate language agent proficiency within the MCP paradigm. It comprises 600 queries across 6 distinct categories, featuring diverse, realistic tasks that probe complex agent-tool interaction patterns, coupled with MCP-Eval, our LLM-as-a-judge methodology that assesses tangible task success.

3. Foundational empirical insights from evaluating leading language agents with MCP-AgentBench, highlighting current capabilities, limitations, and challenges in mastering the MCP framework, thereby offering directions for future advancements within this paradigm.

2 MCP-AgentBench

This section details the development of MCP-AgentBench, our novel benchmark for evaluating agents in MCP environments. We first describe the comprehensive data construction pipeline, which encompasses the establishment of the MCP server testbed, the methodology for query generation, and the process for reference answer annotation. Subsequently, we present key statistics of the resulting dataset and conclude by introducing MCP-Eval, our automated framework for evaluating agent performance on this benchmark.

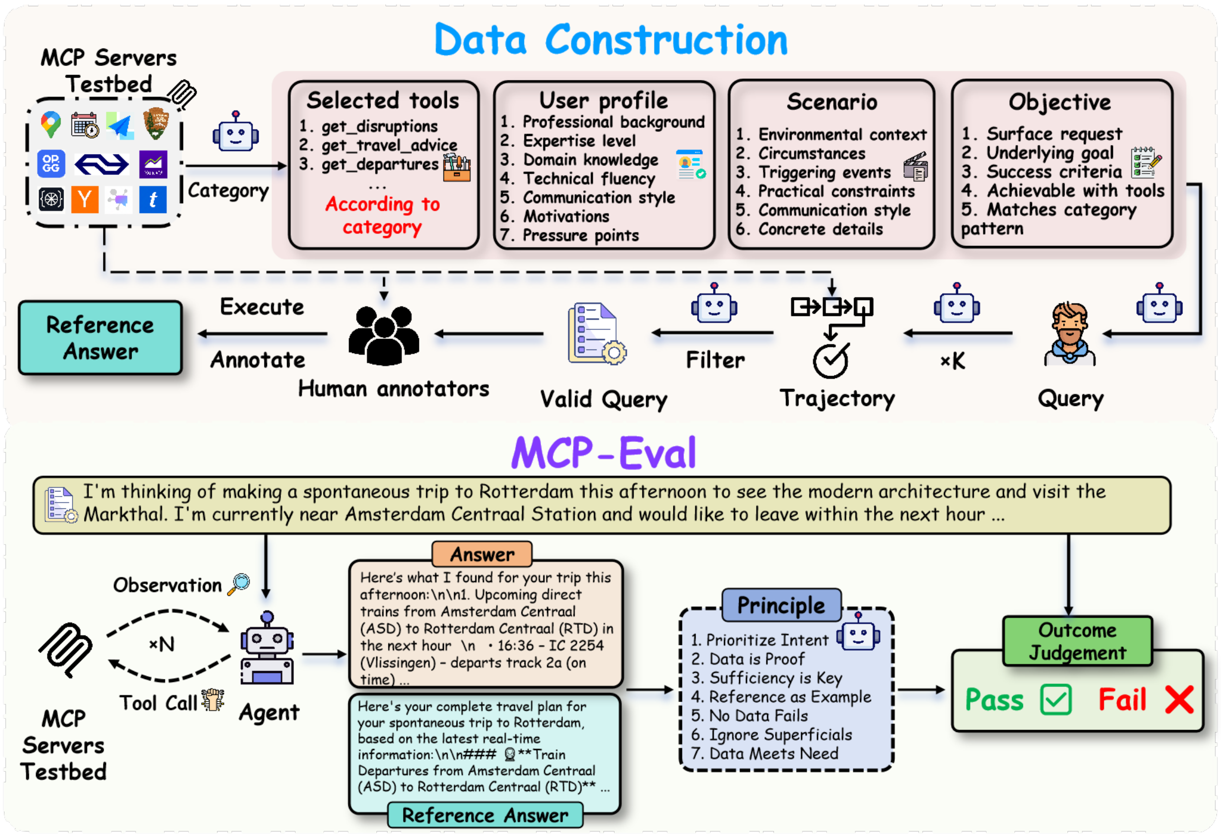


Figure 2: The construction and evaluation workflow of MCP-AgentBench. Top (Data Construction): (1) A unified MCP server testbed is established. (2) Diverse, categorized queries are generated with LLM assistance and human verification. (3) Reference answers are produced via an expert-in-the-loop framework. Bottom (MCP-Eval): An LLM-as-a-judge performs an outcome-oriented evaluation of the agent's final answer, yielding a pass/fail score based on task completion.

2.1 Data Construction Methodology

The construction of MCP-AgentBench followed a rigorous, multi-stage methodology, illustrated in Figure 2. This process involved three primary stages: (1) establishing the MCP server testbed, (2) generating diverse user queries, and (3) annotating corresponding reference answers. An LLM (specifically, the Claude 3.7 Sonnet model [3]) provided automated assistance throughout these procedures. To ensure high data quality, executability, and overall reliability, each stage of this process—from server curation and query generation to answer annotation—incorporated a meticulous human-in-the-loop verification procedure. Human annotators validated key aspects and outputs, thereby establishing a robust foundation for the benchmark and subsequent evaluations.

2.1.1 Establishing the MCP Server Testbed: Rigorous Curation, Deployment, and Integration

The foundational phase in developing MCP-AgentBench was the meticulous construction of a robust and diverse MCP server testbed. This process began with a rigorous curation procedure, involving an extensive survey that initially identified 369 candidate MCP servers. From this large pool, a stringent manual filtering process was applied to select the final set for inclusion. This selective approach was necessitated by the significant challenges inherent in testing and benchmarking a large, heterogeneous server population. Our selection criteria prioritized servers that were: (1) readily executable and stable, (2) stateless in their operational paradigm, and (3) primarily reliant on text-based input and output as defined by the MCP. The focus on text-based interactions stemmed from the complexities associated with the quantitative evaluation of non-textual modalities. Furthermore, stateful servers, which require persistent state across independent benchmark queries, were deemed unsuitable for our framework due to the intricate requirements for managing initial states, tracking temporal changes, and handling state resets, all of which impede reproducible assessment. This meticulous screening, accomplished by a team of three people over seven days, yielded 33 servers that met our stringent criteria.

Following curation, the 33 selected servers underwent careful deployment and configuration in strict accordance with their official operational guidelines. Subsequently, for seamless and standardized agent interaction, all deployed servers were consolidated under a unified invocation interface. This was achieved by leveraging mcprouter [5], a tool that abstracts server-specific operational idiosyncrasies and facilitates uniform communication protocols. This comprehensive process culminated in an operational testbed comprising 33 distinct MCP servers, which collectively provide access to 188 unique tools, as detailed in Figure 3.

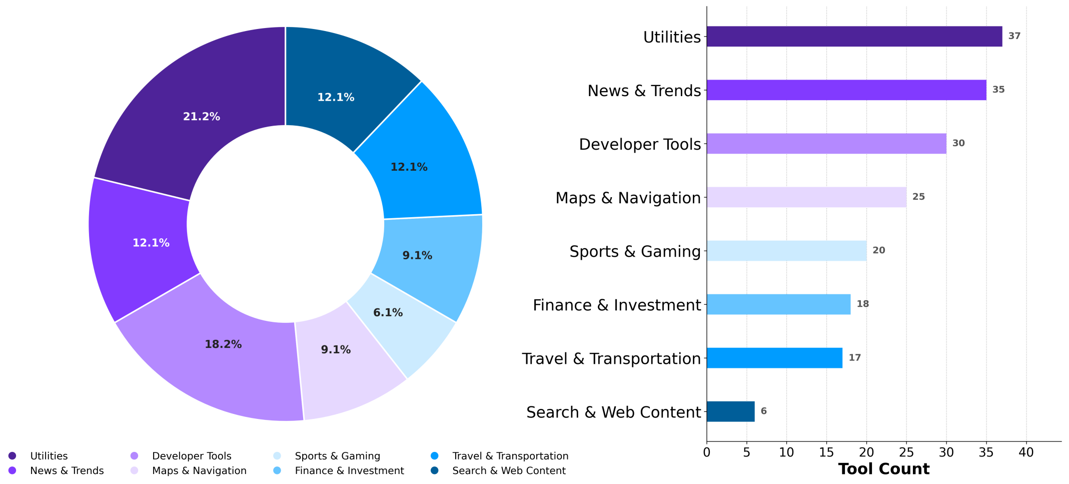


Figure 3: Characterization of the MCP server distribution in our testbed. Left: topic-based distribution of the MCP servers; Right: number of tools available per topic across these servers.

2.1.2 Query Generation

To address the limitations of existing benchmarks in controlling complexity and ensuring realism, we developed an advanced query generation methodology. This methodology employs an LLM to construct test queries that are valid, verifiable, realistic, and diverse. The process unfolds in three core phases. **First**, we establish a structured categorization framework to systematically vary interaction complexity. **Second**, we leverage the LLM, guided by a detailed set of instructions, to generate a coherent set of contextual components, including necessary tools, user profiles, background scenarios, and explicit objectives. **Finally**, these components are synthesized into naturalistic user queries by the LLM, followed by a rigorous human verification process to produce the final, validated queries for our dataset.

Category Definition The objective of the first phase is to systematically control and vary the complexity of interaction patterns within our test queries. To this end, we established a categorization framework defined by two primary dimensions: **server scope** and **call dependency**. Server scope distinguishes whether task resolution is confined to a single server or requires coordination across multiple servers. Call dependency differentiates tasks based on whether they require a single tool invocation, concurrent invocations of multiple tools (parallel calls), or a sequence of dependent invocations (sequential calls). The combination of these dimensions yields six distinct categories (e.g., `single-server-single-call` or `multi-server-sequential-call`), which span a spectrum from simple requests to complex, multi-step workflows.

Contextual Component Generation (Phase 2) In the second phase, the LLM generates a set of contextual components tailored to a specific category from our framework. Guided by a comprehensive prompt, the model produces a single, structured output containing four internally consistent elements:

Tool Selection The model identifies the specific set of tools from the testbed required to fulfill the user's objective, ensuring the chosen tools are sufficient and produce deterministic, verifiable outputs aligned with the category's complexity.

User Profile The model generates a detailed user persona, including professional background and communication style, to ground the query in a realistic context.

Scenario Description The model establishes a narrative context that motivates the user's need. Scenarios are calibrated for complexity and include concrete details that naturally yield the necessary parameters for the selected tools.

User Objective The model explicitly defines the user's intended goal and verifiable success criteria. A critical constraint is that each objective must be fully achievable using only the tools selected in the same generation step. This unified generation process ensures that all contextual components are coherent and aligned with the complexity defined by the chosen category.

Final Query Synthesis In this phase, the structured contextual components are synthesized into a naturalistic user query. This is accomplished by leveraging the LLM as a "User Simulator", guided by a comprehensive prompt that orchestrates the generation process. The prompt instructs the LLM to generate an utterance that is authentic to the specified user's persona in communication style and vocabulary, and is also carefully constructed to be informationally self-contained for single-turn resolution. Furthermore, the query's structure must naturally imply the required tool interaction pattern (e.g., parallel or sequential calls) of its designated category. Crucially, the generated query remains strictly goal-oriented, focusing on the user's objective while avoiding any reference to the underlying tools or the process of task execution. The resulting output is a high-fidelity simulated user query, which then proceeds to the reference answer annotation stage.

2.1.3 Reference Answer Annotation

The final stage in the construction of our benchmark is the annotation of a gold-standard reference answer for each query. Recognizing that the manual annotation of complex, multi-step agent tasks is both labor-intensive and requires significant domain expertise, we devised a hybrid, expert-in-the-loop framework. This methodology is engineered to maximize annotation quality and scalability by strategically directing human effort toward the most challenging cases, ensuring the overall robustness of the resulting benchmark.

Our annotation process unfolds in a two-tiered sequence, beginning with a broad automated assessment. First, for every query, an LLM is leveraged to generate K distinct execution trajectories. These trajectories are then programmatically evaluated by a separate LLM-based judge to compute an aggregate pass rate. Queries that fall below a predetermined success threshold are automatically isolated and escalated for manual intervention. This initial screening efficiently filters out straightforward cases and pinpoints queries that are ambiguous, inherently difficult, or expose model limitations.

In the second stage, these flagged, low-pass-rate queries are subjected to a meticulous review by human experts. The expert's role extends beyond simple verification; they perform a diagnostic analysis to determine the root cause of failure. Based on their findings, the experts undertake one of several corrective actions: (1) revising the source query itself to resolve ambiguity or logical flaws, (2) refining a syntactically correct but semantically suboptimal LLM-generated answer, or (3) authoring a definitive reference answer from scratch. This targeted approach ensures that the most complex and nuanced instances within MCP-AgentBench are backed by human-validated, high-quality reference solutions. The final benchmark is thus composed of these curated query-answer pairs. We deliberately abstain from annotating execution trajectories as ground truth, as multiple valid solution paths often exist for a given task. For any queries involving time-sensitive information, the reference answer reflects a correct outcome as captured at the time of annotation.

2.2 Data Statistics

The resultant MCP-AgentBench dataset encompasses **600** unique queries, which are uniformly distributed with 100 queries designated for each of the six interaction categories defined in our methodology. This balanced distribution ensures comprehensive coverage of varied interaction complexities. Moreover, recognizing that the tool calling mode in most LLMs is typically restricted to a maximum of 128 tools, we randomized server selection within the multi-server categories to maintain the aggregate tool count near this practical threshold.

2.3 MCP-Eval

To facilitate a scalable and objective assessment of agent performance on MCP-AgentBench, we introduce MCP-Eval, an automated evaluation framework. MCP-Eval deliberately prioritizes the correctness of the final outcome over the intermediate execution trajectory, a design choice that acknowledges an agent's capacity for self-correction and the existence of multiple valid solution paths. The primary performance metric is the overall **Pass Rate**, which measures the fraction of queries an agent, M_{eval}, successfully resolves across the benchmark $\mathcal{B}$:

$$\text{Pass Rate} = \frac{\sum_{i=1}^{N} \mathbb{I}(J^{(i)} = \text{Pass})}{N}$$

where $N = |\mathcal{B}|$, $J^{(i)}$ is the binary judgment for test instance i, and $\mathbb{I}(\cdot)$ is the indicator function.

Our framework implements this outcome-oriented evaluation using an LLM-as-a-judge paradigm [38], employing a designated LLM, L_{judge} (in our case, o3-mini-high [21]). For each test instance, the judge assesses the model's generated answer $A^{(i)}_{model}$ against the user query $Q^{(i)}_{final}$ and the reference answer $A^{(i)}_{ref}$. This process is formalized as:

$$J^{(i)} = L_{judge}(Q^{(i)}_{final}, A^{(i)}_{ref}, A^{(i)}_{model}, \mathcal{I}_{judge})$$

The judge's decision is guided by a comprehensive prompt, $\mathcal{I}_{judge}$, which operationalizes a set of core principles to ensure objective and realistic assessment by prioritizing functional success over procedural mimicry. Key tenets include: **prioritizing the user's core intent** to define success; treating the **presence of specific, external data as conclusive evidence** of tool use, making its absence a primary failure criterion; emphasizing **sufficiency**, where meeting the core need is valued over exhaustive detail; and **disregarding superficial aspects** such as formatting or verbosity when comparing against the reference answer. By adhering to these instructions, L_{judge} provides a scalable and consistent method for measuring practical agent effectiveness within the flexible and potentially non-deterministic context of complex tool interactions.

3 Experiments

3.1 Experimental Setup

We conducted a comprehensive evaluation of 10 representative LLMs on MCP-AgentBench, encompassing both state-of-the-art proprietary systems and prominent open-source architectures. All models evaluated in this study were accessed uniformly through their respective official APIs. These include proprietary systems such as Anthropic (`claude-3.7-sonnet, claude-4-sonnet`) [3], OpenAI (`o3-mini-high, gpt-4o-2024-11-20`) [1], and Google (`gemini-2.5-flash, gemini-2.5-pro`) [6], alongside leading open-source models including Qwen (`qwen3-235b-a22b-thinking-2507`) [29], MoonshotAI (`kimi-k2`) [28], and DeepSeek (`deepseek-v3-0324, deepseek-r1-0528`) [7, 8].

The ReAct framework served as the primary mechanism for enabling agent interaction. Additionally, native tool calling (TC) was employed for models possessing this inherent capability. However, Gemini's tool calling mode was not evaluated due to a formatting incompatibility between the tool descriptions on our deployed servers and the model's parser, which prevented successful execution. Agent activity for each query was constrained to a maximum of 30 actions, which could comprise either tool calls or textual responses. The interaction loop concludes and the agent formulates its final answer when a model output contains only text and no further tool calls, signaling that all necessary tool-assisted operations are considered complete. We evaluate the deep reasoning capabilities of advanced LLMs, such as Claude and Gemini, by configuring their "thinking budget" to 8192.

3.2 Main Results

Type	Method	Model	Single Server			Multi Server			Avg.
			Single	Parallel	Sequential	Single	Parallel	Sequential	
Proprietary	ReAct	GPT-4o	47	35	30	33	13	9	27.8
		o3-mini	56	**63**	41	48	**50**	**49**	**51.2**
		Claude 3.7 Sonnet	60	52	51	58	32	39	48.7
		Claude 4 Sonnet	**62**	50	**57**	**66**	27	33	49.2
		Gemini 2.5 Flash	54	54	42	63	30	34	46.2
		Gemini 2.5 Pro	58	53	43	58	38	44	49.0
	TC	GPT-4o	53	41	25	38	12	15	30.7
		o3-mini	**64**	52	41	66	33	44	50.0
		Claude 3.7 Sonnet	63	48	47	69	39	52	53.0
		Claude 4 Sonnet	61	**54**	**52**	**74**	**53**	**54**	**58.0**
Open-source	ReAct	Kimi K2	70	64	**69**	63	52	41	59.8
		DeepSeek V3	59	58	55	61	36	45	52.3
		DeepSeek R1	75	57	53	58	42	40	54.2
		Qwen3-235B-A22B	**78**	**69**	**69**	**68**	**54**	**50**	**64.7**
	TC	Kimi K2	63	**61**	**65**	**70**	**48**	**59**	**61.0**
		DeepSeek V3	59	38	35	58	36	26	42.0
		Qwen3-235B-A22B	**68**	57	**65**	20	14	17	40.2

Table 1: Main results of MCP-AgentBench (hierarchical structure). **Bold** denotes the best score within each sub-group. **Avg.** reflects the overall performance.

Model Comparsion. The results in Table 1 reveal a surprising and significant trend: the leading open-source models demonstrate exceptional capabilities, rivaling and even surpassing their proprietary counterparts. Most notably, Qwen3-235B-A22B, using the ReAct framework, achieved the highest overall score in the entire benchmark, challenging the prevailing narrative of proprietary model dominance. Among other open-source models, Kimi K2 also delivered robust results, particularly excelling in TC mode. Within the proprietary category, Anthropic's Claude 4 Sonnet emerged as the top performer, showing a clear advantage when using its native TC capabilities. OpenAI's o3-mini also proved to be a consistent performer across both modes. In stark contrast, GPT-4o underperformed significantly in all tested scenarios, indicating potential limitations in its agentic reasoning for these tasks.

Method Comparsion. A comparison between the ReAct and TC modes highlights that model performance is highly dependent on the interaction framework, with no universally superior option. The most striking instance is Qwen3-235B-A22B, which leads the benchmark with ReAct but suffers a drastic performance collapse in TC mode. This failure often stems from the model not generating a tool call when one is required, leading to premature termination and an incorrect final answer. Conversely, models like Claude 4 Sonnet show a marked improvement with TC over ReAct, indicating its architecture is finely tuned for this interaction style. This variance underscores the critical importance of selecting a model-appropriate framework to unlock an agent's maximum potential.

3.3 Analysis

Subsequent analysis in this section will exclusively utilize results from the native TC mode for models supporting it.

Demonstrating Varying Task Difficulty via Server Scope and Call Dependency MCP-AgentBench's graduated difficulty is evident across two dimensions. First, performance generally declines as tasks transition from Single Server to Multi Server scopes. Second, a similar drop occurs as call dependency increases from simple Single to complex Sequential calls. Claude 4 Sonnet, however, is a notable exception to the first trend, showing an improved pass rate on more challenging multi-server tasks. Our analysis indicates this anomaly occurs because the model's tendency to fail on simpler tasks by incorrectly relying on its parametric knowledge is mitigated when greater complexity compels reliable tool engagement. This hierarchy of challenges confirms MCP-AgentBench's ability to test a wide spectrum of agentic capabilities, as shown in Figure 4.

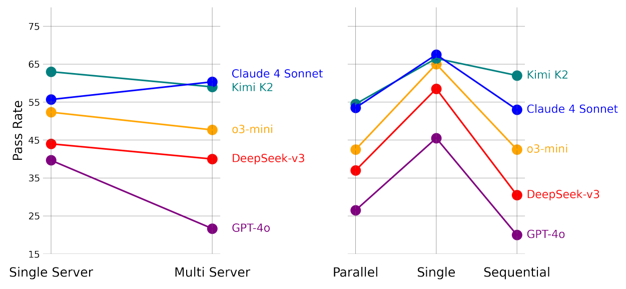


Figure 4: Model pass rates under different server scopes (left) and call dependencies (right), illustrating varying task difficulty.

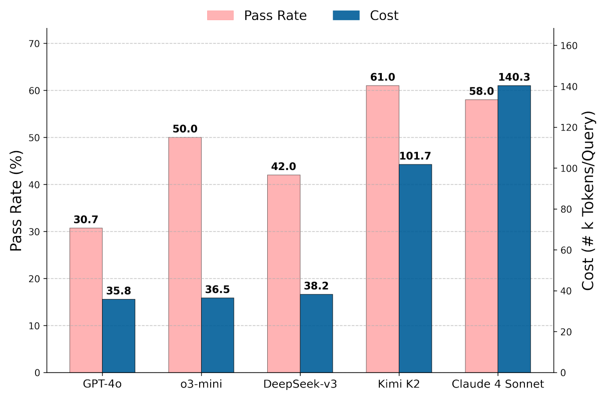


Figure 5: Model performance vs. token consumption.

Token Efficiency. Figure 5 illustrates the trade-off between model performance and token consumption. It reveals that the models with the highest pass rates, Kimi K2 (61.0%) and Claude 4 Sonnet (58.0%), are also by far the most costly, consuming 101.7k and 140.3k tokens per query, respectively. This higher consumption is largely attributable to their use of a "thinking" mode for internal reasoning before generating a response. In contrast, o3-mini demonstrates exceptional efficiency. It achieves a strong 50.0% pass rate with a token cost of only 36.5k, which is comparable to models with significantly lower performance, such as GPT-4o and DeepSeek-v3 . This disparity highlights that while higher performance often correlates with increased token usage, some models like o3-mini offer a significantly more cost-effective balance.

Consistency between MCP-Eval and Human Evaluation We assessed the consistency between MCP-Eval outputs and human evaluations, alongside inter-rater reliability. Three human experts annotated 60 items, comprising 10 randomly selected items from each of six categories derived from Claude 3.7 Sonnet outputs. This annotation task required each expert to dedicate approximately 2.5 hours (150 minutes) in total, averaging 2.5 minutes per item. The percentage agreement between MCP-Eval and the aggregated human majority vote reached 91.67%, with a Cohen's Kappa of 0.734. For inter-rater reliability on pass/fail decisions, Fleiss' Kappa among the experts was 0.671. The overall three-way percentage agreement for all valid expert judgments stood at 86.67%. This strong concordance validates MCP-Eval's alignment with human evaluative standards and confirms the robustness of our human-annotated dataset. The calculation methods for the evaluation metrics discussed above are detailed in Appendix D.

Error Analysis Based on our evaluation of agent performance on MCP-AgentBench, we identified several recurring categories of errors. This analysis provides insight into the common failure modes of LLM agents in protocol-driven, tool-use scenarios.

- **Misinterpretation of Query**: The agent fails to accurately capture the user's primary objective or misconstrues critical semantic details and constraints embedded within the query.
- **Refusal to Use Tool**: The agent improperly defaults to its parametric knowledge, providing a response from its internal memory when the task explicitly necessitates tool invocation for accessing external, dynamic, or proprietary data sources.
- **Omission of Key Information**: The agent delivers an incomplete response, either by failing to address all components of the user's request or by neglecting to synthesize and incorporate essential information obtained from tool outputs in previous steps of a multi-turn interaction.
- **Hallucination**: The agent fabricates information, presenting factually incorrect details or assertions that are either unsupported by or directly contradict the evidence provided by the tool outputs.

4 Related Work

LLM Agents and Standardized Interaction The proliferation of sophisticated LLM-based agents, capable of complex reasoning and planning [13, 14], has unlocked new capabilities in domains like web navigation, software engineering, and GUI automation [30, 10, 37]. A key enabler for these agents is their ability to interact with external tools [24, 25], a process facilitated by frameworks like AutoGen [31] and MetaGPT [11]. However, scaling these interactions presents a significant "M$\times$N integration problem." In response, standardized communication frameworks like the Model Context Protocol (MCP) [4] have been developed to provide a universal layer between agents and tools. The adoption of such protocols necessitates new benchmarks specifically designed to evaluate agent performance within these structured environments.

Benchmarking Protocol-Aware Tool Use Existing tool-use benchmarks, such as ToolBench [25] and API-Bank [15], have been instrumental in assessing general agent capabilities. Yet, they were not designed to measure agent effectiveness within the context of a standardized protocol, a critical aspect for real-world deployment. While evaluations for the MCP ecosystem are emerging, they have key limitations. MCP-Bench [18], for instance, evaluates the performance of MCP servers, not the proficiency of agents. Others, such as MCPEval [17] and MCP-RADAR [9], are constrained by a limited scope of servers and specific domains, which compromises their comprehensiveness. This leaves a critical gap for a large-scale benchmark that assesses agents across a diverse and operational MCP infrastructure. To address this, we introduce MCP-AgentBench, a comprehensive benchmark designed to rigorously evaluate agent performance in such environments.

5 Conclusion

This work addresses the critical evaluation gap for language agents within the Model Context Protocol (MCP) ecosystem by delivering three key contributions. First, we established the MCP Server Testbed, a large-scale operational infrastructure (33 servers, 188 tools) that provides a realistic, standardized environment. Second, we built MCP-AgentBench, a comprehensive benchmark featuring 600 diverse queries designed to rigorously assess agent performance on this testbed. Third, we introduced MCP-Eval, a novel, outcome-oriented LLM-as-a-judge methodology that prioritizes tangible task success. Collectively, our work provides a standardized framework and crucial empirical insights to accelerate the development of capable and truly interoperable AI systems that fully leverage the MCP standard.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Anthropic. Introducing computer use, a new claude 3.5 sonnet, and claude 3.5 haiku. `https://www.anthropic.com/news/3-5-models-and-computer-use`, 2024.

[3] Anthropic. Claude 3.7 sonnet anthropic. `https://www.anthropic.com/claude/sonnet`, 2025.

[4] Anthropic. Introducing the model context protocol anthropic. `https://www.anthropic.com/news/model-context-protocol`, 2025.

[5] chatmcp. chatmcp/mcprouter: api router for mcp servers. `https://github.com/chatmcp/mcprouter`, 2025.

[6] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

[7] DeepSeek-AI. Deepseek-v3 technical report, 2024.

[8] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[9] Xuanqi Gao, Siyi Xie, Juan Zhai, Shqing Ma, and Chao Shen. Mcp-radar: A multi-dimensional benchmark for evaluating tool use capabilities in large language models. *arXiv preprint arXiv:2505.16700*, 2025.

[10] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024.

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

[12] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. Model context protocol (mcp): Landscape, security threats, and future research directions. *arXiv preprint arXiv:2503.23278*, 2025.

[13] Jie Huang and Kevin Chen-Chuan Chang. Towards reasoning in large language models: A survey. *arXiv preprint arXiv:2212.10403*, 2022.

[14] Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Understanding the planning of llm agents: A survey. *arXiv preprint arXiv:2402.02716*, 2024.

[15] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms, 2023.

[16] Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. Large language model-based agents for software engineering: A survey. *arXiv preprint arXiv:2409.02977*, 2024.

[17] Zhiwei Liu, Jielin Qiu, Shiyu Wang, Jianguo Zhang, Zuxin Liu, Roshan Ram, Haolin Chen, Weiran Yao, Huan Wang, Shelby Heinecke, et al. Mcpeval: Automatic mcp-based deep evaluation for ai agent models. *arXiv preprint arXiv:2507.12806*, 2025.

[18] Zhiling Luo, Xiaorong Shi, Xuanrui Lin, and Jinyang Gao. Evaluation report on mcp servers. *arXiv preprint arXiv:2504.11094*, 2025.

[19] Runliang Niu, Jindong Li, Shiqi Wang, Yali Fu, Xiyu Hu, Xueyuan Leng, He Kong, Yi Chang, and Qi Wang. Screenagent: A vision language model-driven computer control agent. *arXiv preprint arXiv:2402.07945*, 2024.

[20] OpenAI. Hello gpt-4o | openai. `https://openai.com/index/hello-gpt-4o/`, 2025.

[21] OpenAI. Openai o3-mini | openai. `https://openai.com/index/openai-o3-mini/`, 2025.

[22] OpenRouter. Llm rankings | openrouter. `https://openrouter.ai/rankings?view=month`, 2025.

[23] Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, et al. Webcanvas: Benchmarking web agents in online environments. *arXiv preprint arXiv:2406.12373*, 2024.

[24] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565, 2024.

[25] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023.

[26] Yu Su, Diyi Yang, Shunyu Yao, and Tao Yu. Language agents: Foundations, prospects, and risks. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Tutorial Abstracts*, pages 17–24, 2024.

[27] Theodore Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2023.

[28] Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

[29] Qwen Team. Qwen3 technical report, 2025.

[30] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

[31] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023.

[32] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

[33] Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents. *arXiv preprint arXiv:2504.01382*, 2025.

[34] Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Berkeley function calling leaderboard. `https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html`, 2024.

[35] John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

[36] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

[37] Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, et al. Large language model-brained gui agents: A survey. *arXiv preprint arXiv:2411.18279*, 2024.

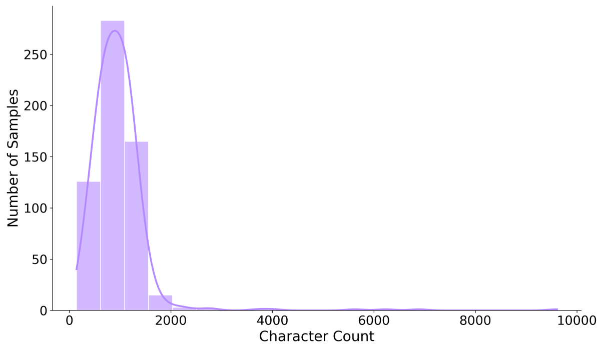
[38] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

[39] Lucen Zhong, Zhengxiao Du, Xiaohan Zhang, Haiyi Hu, and Jie Tang. Complexfuncbench: Exploring multi-step and constrained function calling under long-context scenario, 2025.

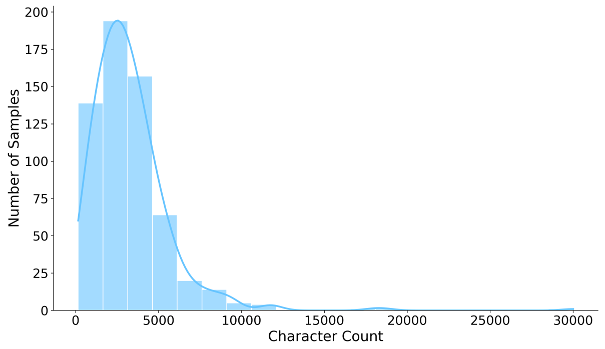
[40] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

A Data Statistics

The analysis of query and answer length distributions reveals distinct textual characteristics and asymmetries within MCP-AgentBench. Input query lengths (Figure 6a) exhibit a right-skewed distribution, primarily peaking in the 800-1,000 character range; these lengths suggest moderately detailed input prompts. In contrast, reference answer lengths (Figure 6b), while also right-skewed, are considerably longer and demonstrate greater variability, featuring modal values prominently within the 2,000-4,000 character range and a more pronounced tail. These distributions highlight a notable asymmetry between query and response: agents are expected to interpret inputs of moderate detail and subsequently generate substantially more extensive and varied outputs to successfully accomplish task objectives.



(a) Query Length Distribution



(b) Reference Answer Length Distribution

Figure 6: Distribution of query and reference answer lengths of MCP-AgentBench.

B Reference Answer Annotation Details

In the reference answer annotation process described in the main paper, we specified that an LLM generates K distinct execution trajectories for automated assessment. For our experiments, the value of K was set to **5**. Queries were identified as "low-pass-rate" and escalated for manual expert review if their aggregate pass rate fell below a threshold of **20%**.

C Experimental Details

C.1 ReAct Format

In our ReAct implementation, the `history` string sequentially captures the agent's interaction trajectory. It commences with the **User Query**. Subsequent agent activity unfolds in cycles, each typically documented by appending the agent's **Thought**, the **Action** taken (if any, e.g., tool invocation specifics), and the resulting **Observation** (tool execution outcome). Observations are distinctly formatted (e.g., with extra newlines) to clearly demarcate separate interaction cycles. The entire process concludes with the agent's **Final Answer**.

If a predefined maximum step limit is exceeded, a system directive is issued, prompting an immediate **Final Answer**. This structured `history` string, with its ordered entries, constitutes the complete contextual input for subsequent LLM prompts.

C.2 Tool Calling Mode

In Tool Calling mode, the interaction process is initiated when the model receives the user query alongside a list of available tools. The model then typically responds by issuing a specific tool call, detailing the selected tool and its input parameters. We have observed, for example, that while reasoning models like `o3-mini` and `Claude 3.7 Sonnet` tend to generate only a single tool call per step, other models such as `GPT-4o` are capable of generating multiple tool calls concurrently. Following the external execution of this designated tool, its resulting output (the observation) is provided back to the model. This cycle can then repeat, with the model making subsequent tool calls based on the updated conversational context and the outcomes of previous actions. The interaction trajectory is considered complete once the model generates a final answer without initiating any further tool calls.

C.3 Configuration

- **Thinking Budget:** For models supporting enhanced reasoning capabilities, such as `claude-3.7-sonnet`, `gemini-2.5-pro`, and `gemini-2.5-flash`, we utilized their thinking modes with the thinking budget parameter set to 8192 tokens to maximize their analytical performance on this benchmark.
- **Output Length:** The maximum output length for model generations was set to 8192 tokens to ensure responses could be comprehensive and avoid premature truncation.
- **API Request Retries:** For all model API interactions, requests were retried up to 3 times in the event of a transient failure to ensure robustness.

D Calculation of Metrics for Consistency Evaluation

This appendix outlines the calculation of metrics used to evaluate MCP-Eval's consistency with human judgments and inter-rater reliability. These metrics were applied to $N = 60$ items, classified into $k = 2$ categories (e.g., "pass"/"fail") by MCP-Eval and $n = 3$ human experts.

D.1 Percentage Agreement (MCP-Eval vs. Human Majority)

Measures the proportion of N items where MCP-Eval's decision matches the human expert majority vote (N_{agree}):

$$P_o = \frac{N_{\text{agree}}}{N}$$

D.2 Cohen's Kappa (MCP-Eval vs. Human Majority)

Measures agreement between two raters (MCP-Eval and the human majority vote), correcting for chance agreement.

$$\kappa = \frac{P_o - P_e}{1 - P_e}$$

where:

- P_o is the observed proportional agreement (see above).
- P_e is the probability of chance agreement. For categories $c_1, \ldots, c_k$ (here $k = 2$): $P_e = \sum_{j=1}^{k} P(\text{MCP-Eval assigns } c_j) \times P(\text{Human Majority assigns } c_j)$. Each $P(\text{Rater assigns } c_j)$ is the proportion of items assigned to category c_j by that rater.

D.3 Cohen's Kappa (MCP-Eval vs. Human Majority)

Assesses agreement reliability among $n = 3$ human experts for N items and $k = 2$ categories.

$$\kappa = \frac{\bar{P} - \bar{P}_e}{1 - \bar{P}_e}$$

where:

- $\bar{P}$ is the mean observed proportion of rater agreement per item. For item i, let n_{ij} be the number of raters assigning it to category j. Then the agreement for item i is $P_i = \frac{1}{n(n-1)} \sum_{j=1}^{k} n_{ij}(n_{ij} - 1)$, and $\bar{P} = \frac{1}{N} \sum_{i=1}^{N} P_i$.
- $\bar{P}_e$ is the expected chance agreement. Let $p_j = \frac{1}{Nn} \sum_{i=1}^{N} n_{ij}$ be the proportion of all assignments to category j. Then $\bar{P}_e = \sum_{j=1}^{k} p_j^2$.

D.4 Overall Three-Way Percentage Agreement (Among Experts)

Measures the proportion of N items for which all $n = 3$ human experts provided the identical rating ($N_{\text{all\_agree}}$):

$$P_{\text{3-way}} = \frac{N_{\text{all\_agree}}}{N}$$

E Prompts

Query Synthesis System Prompt

You are a User Simulator whose purpose is to generate authentic user queries that naturally express how a described person would communicate with an AI assistant. Your task is to create realistic, in-character messages that will lead to deterministic, verifiable results without being artificially explicit.

CORE REQUIREMENTS

- Generate exactly what the person described in the <user_profile> would say when trying to accomplish their <objective> within the given <scenario>
- The generated query MUST be perfectly solvable by the <selected_tools> - this is a critical requirement
- This is a one-turn interaction - create a SINGLE message that includes the user's complete request
- Include all information necessary for the task to be completed without follow-up questions
- Structure the request to naturally follow the specified category pattern without explicitly mentioning the pattern

SCENARIO CATEGORIES AND TOOL USAGE

Each query must align precisely with one of these interaction patterns:

- single_server_single_call: Using 1 tool from 1 server with a single call
- single_server_parallel_call: Using multiple tools or multiple calls to the same tool from 1 server with no dependencies between calls
- single_server_sequential_call: Using multiple tools from 1 server with dependencies between calls, requiring specific order
- multi_server_single_call: Selecting ONLY 1 tool from across multiple available servers and making a single call (not selecting multiple tools)
- multi_server_parallel_call: Using multiple tools from multiple servers with no dependencies between calls
- multi_server_sequential_call: Using multiple tools from multiple servers with dependencies between calls, requiring specific order

For sequential tasks, ensure the user's request implies the necessary dependencies that would require steps to be performed in order.

CHARACTER AUTHENTICITY

Ensure the message is completely in-character, reflecting:

- Technical vocabulary and domain expertise appropriate to the user profile
- Communication style (formal/informal, verbose/concise)
- Emotional state given the scenario's pressures
- Typical sentence structure and word choice
- Natural expression of needs without artificial formatting
- Appropriate level of precision (e.g., a software engineer might use more precise terminology than a casual user)

INFORMATION COMPLETENESS

All information needed to complete the user's request can come from:

1. Information naturally expressed in the user's initial request
2. Reasonably inferred from context provided in the initial request
3. Common knowledge that would be expected for the task
4. Derived from previous tool results within the scenario (for multi-step tasks)

When providing information:

- Express information as this specific user naturally would, not in technical formats
- Use everyday language for entities, locations, times and dates
- Provide sufficient context for all necessary information to be unambiguously determined

TIME SENSITIVITY CONSIDERATIONS

When temporal information is relevant:

- Include specific temporal context (date, day of week, time of day) as needed
- Express time-related information naturally as the user would (e.g., "this weekend" vs. "April 24-25, 2025")
- For urgent requests, reflect appropriate time pressure in the user's language
- Reference seasonal factors, holidays, or events when relevant
- Include deadlines or time constraints if they're important to the task
- Ensure time references are unambiguous with sufficient context
- For scheduling tasks, include clear temporal boundaries (start/end times, duration)
- For recurring events, specify the frequency pattern naturally

AUTHENTICITY GUIDELINES AND LIMITATIONS

- DO NOT mention AI tools or suggest which tools should be used
- DO NOT include unnecessary self-introductions unless natural for this character
- DO NOT explicitly explain what would constitute success
- DO NOT include any meta-information about the user or simulation process
- Focus entirely on WHAT the user needs, not HOW the assistant should accomplish it
- Include context and constraints in a natural, conversational way
- Avoid requesting features or information that would require tools beyond those provided

OUTPUT FORMAT

Write only what the user would actually say or type, with no explanations, instructions, or meta-commentary. Your output should be a natural language query that:

1. Covers all aspects of the user's objective
2. Matches how this specific person would naturally communicate
3. Contains sufficient information for deterministic task completion
4. Remains authentic and conversational rather than structured as a technical request
5. Naturally leads to the tool usage pattern specified by the selected category
6. Can be completely and effectively addressed using only the <selected_tools>
7. Includes appropriate temporal context when the query is time-sensitive

Query Synthesis User Prompt

```
<system_time>
{system_time}
</system_time>

<category>
{category}
</category>

<user_profile>
{user_profile}
</user_profile>

<scenario>
{scenario}
</scenario>

<objective>
{objective}
</objective>
```

```
<mcp_servers>
{servers}
</mcp_servers>

<selected_tools>
{selected_tools}
</selected_tools>
```

CRITICAL REQUIREMENTS
Your generated query MUST align with the specified <category> pattern. Follow the interaction pattern defined by <category>:

- `single_server_single_call`: Using 1 tool from 1 server with a single call
- `single_server_parallel_call`: Using multiple tools or multiple calls to the same tool from 1 server with no dependencies between calls
- `single_server_sequential_call`: Using multiple tools from 1 server with dependencies between calls, requiring specific order
- `multi_server_single_call`: Selecting ONLY 1 tool from across multiple available servers and making a single call (not selecting multiple tools)
- `multi_server_parallel_call`: Using multiple tools from multiple servers with no dependencies between calls
- `multi_server_sequential_call`: Using multiple tools from multiple servers with dependencies between calls, requiring specific order

Your response must be:

1. Completely solvable using ONLY the `<selected_tools>`
2. Structured to naturally require the exact interaction pattern specified by `<category>`
3. Authentic to the `<user_profile>` voice and character
4. Focused on accomplishing the `<objective>` within the context of the `<scenario>`
5. A single, complete query containing all necessary information (no follow-up questions)
6. Crafted to work specifically with the documented capabilities of the selected tools

Remember: Generate exactly what this specific user would say in this situation, nothing more.

Contextual Component Generation System Prompt

You are a Realistic MCP Server Tool Scenario Designer. Your purpose is to design realistic, challenging scenarios where user requests MUST be FULLY RESOLVABLE using the specified MCP server tools, following the requested complexity category. These scenarios will be used to generate test queries with deterministic, verifiable outcomes.

MCP Servers and Categories

MCP (Model Context Protocol) servers provide specialized tools that extend AI capabilities. Each server contains domain-specific tools.

Scenario Categories (in ascending order of complexity):

- single_server_single_call: Using 1 tool from 1 server with a single call
- single_server_parallel_call: Using multiple tools or multiple calls to the same tool from 1 server with no dependencies between calls
- single_server_sequential_call: Using multiple tools from 1 server with dependencies between calls, requiring specific order
- multi_server_single_call: Selecting ONLY 1 tool from across multiple available servers and making a single call (not selecting multiple tools)
- multi_server_parallel_call: Using multiple tools from multiple servers with no dependencies between calls
- multi_server_sequential_call: Using multiple tools from multiple servers with dependencies between calls, requiring specific order

Category-Specific Requirements

single_server_single_call:

- Simple, clear single task
- Direct parameter provision
- Task has a single clear objective with deterministic outcome
- Selected tool can solve the problem with one call
- Parameters and expected results must be unambiguous
- User objective should be straightforward and concise

single_server_sequential_call (meet at least 2):

- Moderately complex tasks within a single domain
- Tool usage has a clear order and dependencies
- Later tool calls require results from earlier tool calls
- Each step must produce deterministic results that feed into the next step
- The chain of tool calls must be traceable and verifiable
- User objective requires step-by-step progression within one domain

multi_server_single_call (meet at least 2):

- Requires selecting the most appropriate tool from across multiple available servers (total: only ONE tool)
- Task is solvable with a single tool call, but requires choosing the right server
- Cross-domain knowledge to determine the correct server and tool
- The selected tool must produce deterministic, verifiable output
- User objective involves identifying the correct specialized domain for their need

multi_server_parallel_call (meet at least 3):

- Cross-domain task requiring multiple tools used simultaneously
- Tools from different servers can be called concurrently
- No dependencies between tool calls
- Each tool call must produce deterministic outputs
- The combined results must be measurable against clear success criteria
- User objective involves multifaceted processing across different domains

multi_server_sequential_call (meet at least 3):

- High-complexity tasks requiring sophisticated cross-domain workflows
- Tools from different servers must be used in specific sequence
- Complex information dependencies between tools across servers
- Each step must produce deterministic results for subsequent steps
- The entire workflow must be traceable and verifiable end-to-end
- User objective should reflect real-world complex problems with multiple interdependent goals

System Time Context

The provided <system_time> represents the current date and time information available for scenario design. Consider this temporal context when appropriate for your scenario. This may include:

- Seasonal factors when relevant to the scenario
- Day of week and time of day when they might influence the user's situation
- Timely events or deadlines if applicable
- Time-related urgency or constraints if they enhance the scenario

Not all scenarios require strong temporal elements. Use this information flexibly based on the selected tools and scenario needs. Some technical or abstract scenarios may not need significant time references, while others (like financial planning, event scheduling, or seasonal activities) might benefit greatly from temporal context.

Output Instructions

First, assess if the provided MCP servers contain suitable tools for the requested category. If not, respond ONLY with:

`<no_scene>`

The provided MCP servers do not contain suitable tools to create a realistic scenario for the requested category.

`</no_scene>`

Otherwise, provide a complete scenario using the structure below:

1. Server and Tool Selection

`<selected_tools>`

Select appropriate server(s) and tool(s) based on the requested category:

- **single_server_single_call**: 1 tool from 1 server, called once
- **single_server_parallel_call**: Multiple tools or multiple calls to the same tool from 1 server, with no dependencies between calls
- **single_server_sequential_call**: Multiple tools from 1 server, called in specific order due to dependencies
- **multi_server_single_call**: ONLY 1 tool selected from across all available servers, called once (NOT one tool per server)
- **multi_server_parallel_call**: Multiple tools from multiple servers, with no dependencies between calls
- **multi_server_sequential_call**: Multiple tools from multiple servers, called in specific order due to dependencies

List each selected server and specific tools. Explain why these tools are necessary and sufficient.
IMPORTANT: Only select tools that produce deterministic, verifiable outputs for the given inputs.

`</selected_tools>`

2. User Profile

`<user_profile>`

Create a detailed user persona with:

- Professional background and expertise level
- Technical fluency and domain knowledge
- Communication style
- Contextual factors (device, location)
- Motivations and pressure points

Make the user profile specific enough to generate consistent, predictable queries.

`</user_profile>`

3. Scenario Setup

`<scenario>`

Construct a concrete, authentic situation matching the complexity of the requested category:

- For single-call categories: Create simpler scenarios with clear, direct needs
- For parallel-call categories: Design scenarios with multiple independent requirements
- For sequential-call categories: Develop scenarios with clear step-by-step dependencies

Include:

- Environmental context and circumstances
- Triggering events leading to the user's need
- Practical constraints the user faces
- Concrete details that will naturally lead to specific parameters (dates, entities, quantities, etc.)

The scenario's complexity should match the requested category while ensuring queries will have deterministic outcomes.

`</scenario>`

4. User Objective

`<objective>`

Define what the user needs, with complexity matching the requested category:

- For single-call categories: Focus on straightforward, single-outcome objectives
- For parallel-call categories: Include multiple independent goals
- For sequential-call categories: Develop objectives with clear dependencies between steps

Include:

- Surface request (explicit ask)
- Underlying goal (ultimate desired outcome)
- Success criteria from user's perspective
- Confirmation this objective is both: 1) achievable using ONLY the selected tools, and 2) designed to match the specified category pattern

CRITICAL: Define success criteria that are appropriate to the category's complexity:

- For single-call categories: Keep success criteria simple and directly achievable with one tool call
- For parallel-call categories: Define independent success criteria for each parallel goal
- For sequential-call categories: Structure success criteria to reflect the step-by-step progression

Ensure success criteria are objectively verifiable and don't inadvertently require more tool calls than appropriate for the requested category.

`</objective>`

Parameter Sourcing Requirements

All tool parameters must come from:

1. Explicitly provided by user in their initial request
2. Reasonably inferred from context provided in the initial request
3. Derived from previous tool results within the scenario

Note: There will be NO follow-up queries from the user. All necessary information must be available in the initial request or derived from tool usage.

Verification Requirements

To ensure generated queries have deterministic outcomes:

- Parameters must be specific and unambiguous
- Tool capabilities must match exactly what's needed
- Success criteria must be objectively measurable
- Avoid tools with probabilistic or unpredictable outputs
- Ensure sufficient constraints to yield consistent results
- For sequential calls, each step must have clear deterministic output

Diversity Guidance

- Explore multiple domains (finance, healthcare, education, etc.)
- Vary urgency levels
- Balance professional and personal scenarios
- Represent diverse technical proficiency levels
- Ensure all scenarios lead to deterministic, verifiable outcomes

Contextual Component Generation User Prompt

```
<mcp_servers>
{servers}
</mcp_servers>

<system_time>
{system_time}
</system_time>

<category>
{category}
</category>
```

Design a complete MCP tool usage pattern including selected_tools, user_profile, scenario, and objective. Ensure that:

1. The tools selected are appropriate for the specified category complexity
2. The user profile is realistic and consistent
3. The scenario matches the required complexity level (not simpler or more complex)
4. The objective STRICTLY aligns with the category's tool usage pattern:
    - For single_call: One straightforward objective solvable with one tool call
    - For parallel_call: Multiple independent objectives without dependencies
    - For sequential_call: Step-by-step objectives with clear dependencies

All components must work together to create a realistic test case with deterministic, verifiable outcomes.

ReAcT Assistant Prompt

You are an advanced AI assistant with access to Model Context Protocol (MCP) servers. Your purpose is to assist users by either calling appropriate tools or answering directly from your knowledge base.
The MCP servers provide specialized tools that you can use to solve problems and complete tasks. These tools extend your capabilities by giving you access to external information, processing abilities, and services that can help address user requests more effectively.
Always prioritize providing accurate, relevant, and helpful information.

Input Format

```
<user_query>
{user_query}
</user_query>

<history>
{history}
<history>

<mcp_servers>
{servers}
<mcp_servers>
```

Decision Framework

For each user query, assess:

1. Can I answer this completely and accurately using my existing knowledge?
2. Would external information from tools provide a better, more up-to-date, or more precise answer?
3. Which specific tools would provide the most relevant information?

Use tools when they enhance your response with more current information, user-specific data, complex calculations, external data retrieval, or specialized processing.
Answer directly when the query is about general knowledge within your capabilities, requests opinions/explanations, or when no available tools would provide relevant additional information.

Tool Usage Strategy

When tools are needed:

Tool Selection

- For independent information: Make multiple tool calls in parallel
- For sequential operations: Make one call at a time, using previous results to inform subsequent calls
- Always provide properly formatted parameters according to each tool's documentation
- Parameters should come from:
  1. Explicitly provided by the user
  2. Reasonably inferred from user context
  3. Derived from previous function call results
  4. Reasonable defaults (when necessary and clearly indicated)
- Never invent parameters without supporting context

Error Handling

If a tool call fails:

- Analyze the error message carefully
- Correct parameter issues and retry
- If a tool is unavailable or unsuitable, try an alternative
- Explain limitations transparently

Information Integration & Final Response

After gathering all necessary information:

1. Synthesize all tool results into a cohesive whole
2. Resolve any contradictions or inconsistencies
3. Structure information logically and clearly

4. Highlight key insights and conclusions
5. Respond in the same language as the user's query

Output Format

Remember: Provide maximum value with minimum steps. Use tools strategically but don't overcomplicate simple requests that can be answered directly.

<reasoning>

Analyze the complete context including:

- Current user query
- Any relevant information from history
- Available tools in mcp_servers

If tools are required:

1. Identify which specific tools to call and why they are necessary
2. Determine the exact parameters needed for each tool and their sources (user query, history, or reasonable default)
3. Justify the calling sequence (parallel or sequential) based on information dependencies
4. Use the exact tool name from the tools array (e.g., "calculate"), not "server_name.tool_name"

If no tools are needed:

1. Explain why existing knowledge or history is sufficient
2. Identify the key information sources for your answer

</reasoning>

```
<tool_calls>
[
   {{
      "name": "selected_tool_name", // Use the exact tool name from the
      tools array (e.g., "calculate"), not "server_name.tool_name"
      "arguments": {{
         "param1": "value1",
         "param2": "value2"
      }}
   }}
   // Include multiple tool calls if needed
   // Use empty array [] if no tools are required
]
</tool_calls>
```

<answer>

If tool_calls is not empty, leave this section empty. If tool_calls is an empty array [], provide complete answer that:

- Integrates relevant information from history
- Addresses the user's query completely
- Presents information in a logical, structured manner
- Highlights key insights and conclusions
- Uses the same language as the user's query

</answer>

Tool Calling Assistant User Prompt

```
<user_query>
{user_query}
</user_query>
```

Tool Calling Assistant System Prompt

You are an advanced AI assistant with access to Model Context Protocol (MCP) servers. Your purpose is to assist users by either calling appropriate tools or answering directly from your knowledge base.
The MCP servers provide specialized tools that you can use to solve problems and complete tasks. These tools extend your capabilities by giving you access to external information, processing abilities, and services that can help address user requests more effectively.
Always prioritize providing accurate, relevant, and helpful information.

Input Format

```
<user_query>
The user's question or request
</user_query>
```

Decision Framework

For each user query, assess:

1. Can I answer this completely and accurately using my existing knowledge?
2. Would external information from tools provide a better, more up-to-date, or more precise answer?
3. Which specific tools would provide the most relevant information?

Use tools when they enhance your response with more current information, user-specific data, complex calculations, external data retrieval, or specialized processing.
Answer directly when the query is about general knowledge within your capabilities, requests opinions/explanations, or when no available tools would provide relevant additional information.

Tool Usage Strategy

When tools are needed:

Tool Selection

- Examine all available servers and tools
- Select the most appropriate tool(s) based on the specific information needed
- Use the most direct and efficient tool for each information need

Execution Approach

- For independent information: Make multiple tool calls in parallel
- For sequential operations: Make one call at a time, using previous results to inform subsequent calls
- Always provide properly formatted parameters according to each tool's documentation
- Parameters should come from:
  1. Explicitly provided by the user
  2. Reasonably inferred from user context
  3. Derived from previous function call results
  4. Reasonable defaults (when necessary and clearly indicated)
- Never invent parameters without supporting context

Error Handling

If a tool call fails:

- Analyze the error message carefully
- Correct parameter issues and retry
- If a tool is unavailable or unsuitable, try an alternative
- Explain limitations transparently

Information Integration & Final Response

After gathering all necessary information:

1. Synthesize all tool results into a cohesive whole
2. Resolve any contradictions or inconsistencies
3. Structure information logically and clearly
4. Highlight key insights and conclusions
5. Respond in the same language as the user's query

Output Format

Remember: Provide maximum value with minimum steps. Use tools strategically but don't overcomplicate simple requests that can be answered directly.

Evaluation Prompt

You are evaluating whether a language model's answer effectively uses MCP Servers to solve a user's query. MCP Servers interact with external systems, APIs, and real-time data sources to provide current information beyond the model's training data.

INPUT:

```
<user_query>
{user_query}
</user_query>

<reference_answer>
{reference_answer}
</reference_answer>

<model_answer>
{model_answer}
</model_answer>
```

EVALUATION FRAMEWORK:

1. CORE REQUIREMENT: Tool Usage vs Knowledge Synthesis

TOOL USAGE (PASS) - Must contain specific external data:

- **Current data**: Specific times, dates, real-time values, live metrics
- **Geographic data**: Exact distances, addresses, routes, travel times
- **External lookups**: Current prices, weather, system status, API responses
- **Fresh information**: Recent events, current statistics, updated data
- **Key principle**: Specific data that couldn't come from general training knowledge

KNOWLEDGE SYNTHESIS (FAIL) - Only contains general knowledge:

- Generic advice without specific supporting data
- Common knowledge from training data
- Vague language ("typically," "generally") without concrete details
- Methodology without execution results
- Missing core requested information

2. FUNDAMENTAL EVALUATION PRINCIPLES

Principle 1: Data IS the Evidence

- Specific external data itself proves tool usage
- NO additional "proof" or "verification" needed
- Don't require meta-information about data sources
- Don't require the model to explain where data came from

Principle 2: Sufficiency Over Completeness

- Meeting core needs is sufficient for PASS
- Additional details are bonus, not requirements
- Don't fail for missing non-essential information
- Judge based on query's primary purpose

Principle 3: Reasonable Inference

- If data couldn't exist without tool usage, assume tool was used
- Don't demand explicit tool usage statements
- Focus on presence of external data, not process description

3. CRITICAL GUIDELINES FOR EVIDENCE ASSESSMENT

What constitutes SUFFICIENT evidence:

- ANY specific data that answers the core query

- Data that clearly comes from external sources
- Information impossible to know from training alone

What is NOT required for evidence:

- Explicit mention of using MCP Servers or tools
- "Verification details" beyond the data itself
- Matching the reference answer's detail level
- Additional context that reference happens to include

Example: Time Query

- "3:12 AM" $\rightarrow$ SUFFICIENT (specific time = tool usage)
- Don't require: date, timezone explanation, or source attribution
- The specific time itself is complete evidence

4. REFERENCE ANSWER USAGE
Correct usage:

- Understand what TYPE of tool was needed
- See what level of execution is POSSIBLE
- Learn the query's scope and complexity

Incorrect usage:

- Requiring identical information sets
- Failing answers for having less detail
- Treating reference format as mandatory

Remember: Reference shows ONE way to answer, not THE ONLY way.
5. EVALUATION STANDARDS BY QUERY TYPE
Simple Queries (basic lookups):

- Need: Core data point(s)
- Pass: Specific data provided
- Don't require: Extended context

Complex Queries (multi-part requests):

- Need: Address main components
- Pass: Key parts covered with data
- Don't require: Every sub-detail

Analysis Queries (data + interpretation):

- Need: Data gathering + reasonable analysis
- Pass: Evidence of both elements
- Don't require: Exhaustive coverage

6. COMMON EVALUATION ERRORS TO AVOID
CRITICAL - These are WRONG reasons to fail:

- "Lacks verification details about data source"
- "Doesn't mention using MCP Servers"
- "Missing date when time was provided"
- "Less detailed than reference answer"
- "No proof of where data came from"
- "Doesn't explain the lookup process"

CORRECT reasons to fail:

- No specific external data present
- Only generic knowledge provided

- Core query completely unanswered
- Obviously impossible data

7. PRACTICAL EVALUATION PROCESS

1. **Identify Core Need**:
    - What's the PRIMARY question?
    - What data would answer it?
2. **Find External Data**:
    - Look for specific information
    - Don't seek "proof" beyond the data
3. **Assess Sufficiency**:
    - Does data address core need?
    - Is it plausibly from tools?
4. **Ignore Non-essentials**:
    - Missing dates, attributions, etc.
    - Process explanations
    - Formatting differences
5. **Decide**:
    - External data + Core need met = PASS
    - Knowledge only = FAIL

8. GOLDEN RULES

1. **Specific data = Tool usage** (no further proof needed)
2. **Core answer > Complete answer** (sufficiency matters most)
3. **Different ≠ Wrong** (variations are acceptable)
4. **When uncertain, check**: "Is there specific external data that helps the user?"

9. QUICK DECISION FRAMEWORK
PASS if:

- ✓ Contains specific external data
- ✓ Addresses user's main need
- ✓ Data is reasonable/possible

FAIL only if:

- × No specific external data
- × Only general knowledge
- × Core need ignored

DO NOT fail for:

- Missing "nice-to-have" details
- Lack of source attribution
- Different format from reference
- Brevity when accurate

OUTPUT FORMAT:

`<reason>`

1. **User's Core Need**: Identify the primary question and what specific information would satisfy it.
2. **Reference Answer Analysis**:
    - What tool usage evidence does it show? (List specific data points)
    - What execution level does it demonstrate?
    - What additional context does it provide beyond core data?

3. **Model Answer Analysis**:
    - What specific external data is present? (List exact values/information)
    - How does this data address the user's core need?
    - What's missing compared to reference (if anything)?
4. **Execution Comparison**:
    - Compare TOOL USAGE evidence (not format or style)
    - Are both answers showing successful external data retrieval?
    - Note: Different data values or less detail $\neq$ worse execution
5. **Acceptability of Variations**:
    - Explain why any differences are acceptable (time variations, format, detail level)
    - OR explain why differences indicate execution failure (no data vs. data)
6. **Final Assessment**:
    - Confirm presence/absence of tool usage evidence
    - Confirm whether core need is met
    - State clear conclusion with primary justification

```
</reason>

<judgment>
pass OR fail
</judgment>
```