scale

MCP-Atlas: A Large-Scale Benchmark for Tool-Use Competency with Real MCP Servers

Chaithanya Bandi[1], **Ben Hertzberg**[1], **Geobio Boo**[1], **Tejas Polakam**[1], **Jeff Da**[1], **Sami Hassaan**[1], **Manasi Sharma**[1], **Andrew Park**[1], **Ernesto Hernandez**[1], **Dan Rambado**[1], **Ivan Salazar**[1], **Rafael Cruz**[1], **Chetan Rane**[1], **Ben Levin**[1], **Brad Kenstler**[1], **Bing Liu**[1]

[1]Scale AI

Abstract

The Model Context Protocol (MCP) is rapidly becoming the standard interface for Large Language Models (LLMs) to discover and invoke external tools. However, existing evaluations often fail to capture the complexity of real-world scenarios, relying on restricted toolsets, simplistic workflows, or subjective LLM-as-a-judge metrics. We introduce MCP-Atlas, a large-scale benchmark for evaluating tool-use competency, comprising 36 real MCP servers and 220 tools. It includes 1,000 tasks designed to assess tool-use competency in realistic, multi-step workflows. Tasks use natural language prompts that avoid naming specific tools or servers, requiring agents to identify and orchestrate 3-6 tool calls across multiple servers. We score tasks using a claims-based rubric that awards partial credit based on the factual claims satisfied in the model's final answer, complemented by internal diagnostics on tool discovery, parameterization, syntax, error recovery, and efficiency. Evaluation results on frontier models reveal that top models achieve pass rates exceeding 50%, with primary failures arising from inadequate tool usage and task understanding. We release the task schema, containerized harness, and a 500-task public subset of the benchmark dataset to facilitate reproducible comparisons and advance the development of robust, tool-augmented agents.

1. Introduction

Large Language Models (LLMs) are increasingly deployed as agents capable of planning, invoking, and integrating external tools to achieve complex user objectives. The Model Context Protocol (MCP) has emerged as a critical substrate for these agents, standardizing the mechanisms for server discovery, tool enumeration, typed parameter passing, and result consumption. Despite its growing adoption, the credible evaluation of MCP-capable agents remains a significant challenge.

Practical deployments demand evaluations that incorporate: (i) realistic, multi-step workflows involving branching logic and cross-server orchestration; (ii) breadth across authentic MCP servers and APIs, rather than mock servers; and (iii) objective, reproducible scoring mechanisms disentangled from agent verbosity or stylistic preferences. Existing MCP-based benchmarks have made strides but exhibit critical limitations in scale, evaluation methodology, or task complexity.

We introduce MCP-Atlas, a large-scale, real-server benchmark designed to measure tool-use competency under realistic single-turn prompts that necessitate complex, multi-tool workflows. We evaluate using a claims-based rubric: each task defines a set of factual claims that the correct answer must contain, enabling objective partial-credit scoring. This approach decouples success from adherence to a specific trajectory while capturing rich diagnostics on discovery, parameterization, and recovery behaviors.

MCP-Atlas features an environment that exposes a limited tool set per task, mixing required tools with plausible distractors to test the agent's ability to identify and select the correct tools. The full data set includes 1,000 tasks spanning 36 servers and 220 tools. The prompts elicit 3-6 tool calls; notably, the vast majority of tasks require orchestration across multiple servers, and a third of the tasks involve conditional branching.

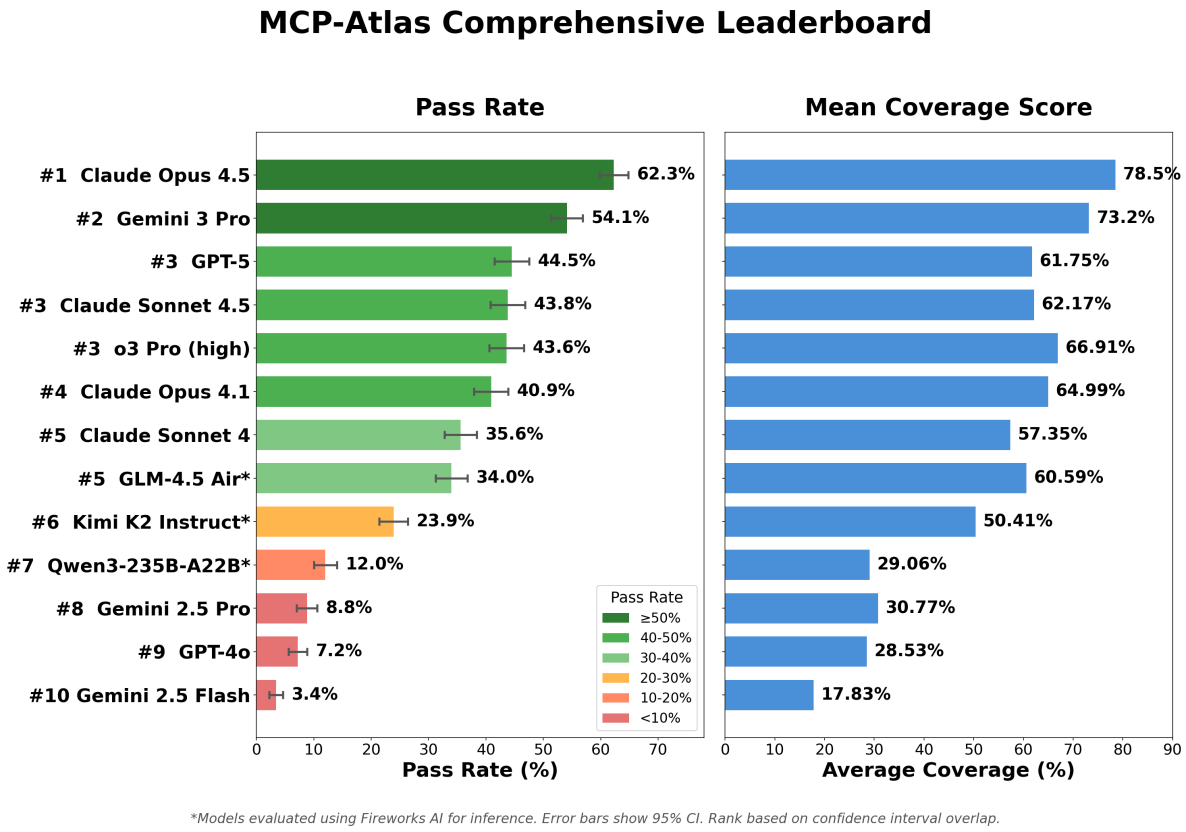


Figure 1: MCP-Atlas overall model performance ranked by pass rate on the 1,000 task set. The best-performing model (Claude Opus 4.5) achieves 62.3% success, highlighting progress while showing room for improvement on multi-step, real-world tool use tasks.

Unlike purely execution-based suites, MCP-Atlas uses the final answer for primary pass/fail evaluation via a claims-based rubric. Each task specifies a set of factual claims the correct answer must contain. Non-blocking diagnostics capture tool selection, typing, parameterization, errors, and efficiency to characterize failure modes (See §4.1 for the rubric and §4.3 for harness details).

Experiment results reveal that top models achieve pass rates exceeding 50% while next-best models score in the 20-40% range, with high variance. Errors predominantly cluster in TOOL USAGE (incorrect server selection, parameter errors, sequencing mistakes) and TASK UNDERSTANDING (premature stopping, missed subgoals). These patterns align with the "unknown-tools" difficulties reported in related work [15, 17].

Contributions. Our main contributions are summarized as follows:

- **A Large-Scale, Realistic Benchmark.** We introduce MCP-Atlas, a real-server MCP benchmark with 1,000 tasks covering 36 servers and 220 tools. Tasks use natural language prompts that avoid naming tools and reliably elicit complex behaviors, including 3-6 tool calls per task, with the vast majority of tasks requiring cross-server composition, and approximately one-third involving conditional branching.
- **A Public Release of Data.** We open source a 500-task subset from the MCP-Atlas benchmark data. The remaining 500 tasks are retained as a held-out validation set to prevent overfitting and enable future leaderboard integrity. The public set closely represents the entirety of the benchmark, making use of 36 servers, 220 tools, and maintaining 3-6 tool calls per task.
- **Claims-Based Scoring with Internal Diagnostics.** We employ a claims-based rubric with partial credit for primary pass/fail evaluation, reducing subjectivity compared to holistic LLM-as-judge approaches by decomposing evaluation into independent, verifiable claims. This is supplemented by internal diagnostics (not publicly released) capturing discovery, parameter/type correctness, error recovery, and efficiency (detailed in Appendix A).

- **Reproducible Evaluation Harness and Baselines.** We provide a containerized harness featuring servers, allow-listed egress, and controlled tool exposure with distractors. We release the schemas to ensure reproducible comparisons. Baselines highlight substantial headroom for improvement in current LLM capabilities, reinforcing findings on the friction associated with "unknown-tools" [15].

2. Related Work

2.1 The Evolution from Static to Agentic Evaluation

The evaluation landscape for LLMs has shifted significantly from static, knowledge-centric assessments to dynamic, interactive paradigms that measure agentic capabilities. Early benchmarks, such as MMLU [1] and HELM [2], focused on fixed-question sets evaluating factual recall and reasoning in isolated settings. While foundational, these benchmarks fail to capture the demands of real-world deployment, where models must interact with environments, manage uncertainty, and adapt plans dynamically.

Subsequent work introduced interactive evaluations emphasizing tool use and environmental interaction. Web-based benchmarks like WebArena [3] and MiniWoB++ [4] require agents to navigate and manipulate GUIs for tasks such as information retrieval or online shopping. OS-level suites, including OSWorld [5] and Android Arena [6], extend this to operating systems, testing multimodal control and long-horizon planning.

API- and function-calling benchmarks further advanced the field. ToolBench [7] aggregates thousands of RESTful APIs for multi-step tasks. BFCL [9] provides leaderboards focusing on function-calling with synthetic and real APIs. τ-Bench [10] emphasizes retail, airline, and telecommunications domains with interleaved user-agent interactions. Benchmarks like SWE-Bench [11] (software engineering) and GAIA [12] (general assistance) incorporate diverse tools (code execution, web search), highlighting challenges in planning, error recovery, and orchestration.

2.2 MCP-Based Evaluation

The Model Context Protocol (MCP) [13, 14] standardized tool declarations and server composition, spurring benchmarks that better reflect production environments. Table 1 summarizes existing work. While these efforts have advanced the field, they reveal a persistent tension: benchmarks optimizing for evaluation rigor tend to remain small, while those achieving scale often compromise on objectivity or realism. MCP-Atlas is designed to resolve this tension.

Early MCP benchmarks prioritized rigorous, programmatic evaluation. MCP-Universe [15] introduced format, static, and dynamic evaluators with real-time ground truth verification, identifying the critical *unknown-tools* challenge that motivates our distractor design. Toolathlon [21] and MCPMark [22] similarly employ dedicated verification scripts against live software environments. However, the cost of manual task creation and custom verifiers limits these benchmarks to under 250 tasks—insufficient for statistically robust comparisons across the diversity of tool-use scenarios agents encounter in practice.

Efforts to scale have introduced different trade-offs. MCPEval [17] reaches 676 tasks through automated generation, but this can compromise task quality and naturalism. MCP-Bench [16] and LiveMCPBench [18] adopt holistic LLM-as-judge scoring, which enables scale but introduces style biases—verbose responses may score differently than terse correct answers, reducing reproducibility. MCPVerse [23] and MCP-RADAR [19] expand server coverage but rely partly on synthetic or mock implementations, limiting how well results predict real-world performance.

MCP-Atlas resolves these trade-offs through three design choices. First, we achieve scale (1,000 tasks across 36 servers and 220 tools) while maintaining quality through systematic manual verification rather than automated generation. Second, we replace holistic LLM-as-judge scoring with *claims-based evaluation*: each task defines independent, verifiable factual claims that the correct answer must contain, enabling objective partial-credit scoring without the authoring burden of full programmatic validation (achieving 78% agreement with human judges; see §4.1). Third, we ensure evaluation fidelity by using exclusively real MCP servers and systematically including 5–10 plausible distractors per task, directly testing the tool discovery capabilities that prior work identifies as a primary failure mode. Together, these choices yield a benchmark that is simultaneously large-scale, objective, and realistic.

Table 1: Positioning MCP-Atlas among contemporary MCP evaluations. Scale metrics include number of servers (#S), tools (#T), and tasks (#Tasks). For Distractors column: ✓ = systematic distractors included, blank = no distractors.

Benchmark	#S	#T	#Tasks	Cross-Server	Distractors	Natural Lang.	Real Servers
MCP-Universe [15]	11	133	231	✓	✓	✓	✓
MCPEval [17]	5	19	676	Partial		Partial	✓
MCP-Bench [16]	28	250	104	✓	✓	✓	✓
Toolathlon [21]	32	604	108	✓	✓	✓	✓
MCPMark [22]	5	∼50	127			✓	✓
LiveMCPBench [18]	70	527	95	✓	✓	✓	✓
MCPVerse [23]	65	552	–	✓	✓	✓	Mixed
MCP-RADAR [19]	42	–	507	Partial		Mixed	Mixed
MCP-Atlas (Ours)	**36**	**220**	**1,000**	✓	✓	✓	✓

3. Benchmark Design and Overview

MCP-Atlas is designed to evaluate LLM agents on realistic, single-turn tasks requiring multi-tool orchestration over live MCP servers. By constraining the interaction to a single user prompt while eliciting a range of 3-6 tool calls, the benchmark emphasizes tool discovery, precise parameterization, cross-server coordination, and grounding in tool outputs. Tasks are executed in a controlled environment with partial tool exposure, including systematic distractors, to probe the agent's ability to select and sequence appropriate tools without explicit guidance.

3.1 Scope and Scale

MCP-Atlas provides 1000 tasks to analyze the ability of models to handle multi tool processes. Each task is designed to demand multi-step workflows, typically requiring 3-6 tool calls, with the vast majority of tasks requiring 2 or more distinct MCP servers, and approximately one-third of the tasks incorporating conditional logic. Conditional logic refers to tasks where the choice of subsequent tool calls depends on the output of earlier calls (e.g., querying a fallback search engine if the primary yields insufficient results), as opposed to tasks with predetermined tool sequences where all required tools are known upfront.

The benchmark spans 36 real MCP servers and 220 tools across diverse domains, including search, data analytics, productivity, finance, and coding (see Table 1). **Critically, all servers are real, production MCP implementations**–not synthetic mocks or simulated environments–ensuring that evaluation results reflect actual deployment behavior including real API responses, rate limits, and error conditions. This breadth ensures coverage of production-like scenarios. Servers are hosted in an isolated container, mirroring real-world security constraints (e.g., sandboxed file systems and allow-listed network egress). Unlike narrower benchmarks [17, 19], the scale of MCP-Atlas enables wider evaluation of tool usage and complex cross-domain orchestration, such as integrating financial APIs with news retrieval for market analysis.

3.2 Task Schema

Each task in MCP-Atlas is defined by four mandatory components, ensuring consistency, solvability, and objective scoring:

1. **Tool Set Configuration:** A controlled subset of 10-25 tools exposed to the agent per task. This comprises 3-7 target tools essential for the reference solution and 5-10 distractors from similar categories. This setup probes tool discovery precision and recall, discouraging brute-force enumeration. Target tools are required to be sourced from 2 or more distinct servers.

2. **Prompt:** A single-turn, natural-language request requiring multiple tool calls. Prompts are designed to be tool-dependent (unsolvable via LLM knowledge alone), time-invariant, unambiguous, and conversational. Critically,

Table 2: Environment buckets with representative servers and recurrent pitfalls captured by diagnostics.

Bucket	Representative Servers (Examples)	Typical Pitfalls / Skills Probed
BASIC	brave_search, ddg_search, exa, calculator, weather, google-maps	Query formulation; date/geo formats; pagination; disambiguating near-synonyms (local vs. web search).
ANALYTICS	airtable, mongodb, f1-mcp-server	Schema alignment; filter/aggregation parameters; type correctness; joins.
PRODUCTIVITY	filesystem, notion, slack, arxiv, google-workspace, pubmed	Path scoping and sandboxing; ID-vs-keyword retrieval; rate/size limits on document fetch.
FINANCIAL	twelvedata, alchemy	Window alignment (market hours, time zones); symbol scoping; split/adjusted series handling.
CODING	git, github, mcp-code-executor, cli-mcp-server	Stateful repo operations (branch/checkout); diff scope; command argument validation.

they employ natural language instructions that avoid explicit mentions of server or tool names and obvious parameter hints (e.g., quoted keywords) to test the agent's reasoning and discovery capabilities. Prompts prioritize multi-hop reasoning, requiring outputs from initial tools to guide later tool calls, and inference of implicit conditions.

3. **Reference Trajectory:** The minimal sequence of tool calls (names, methods, dependencies, arguments, and tool outputs) that resolves the task, including conditional branch points. This is used strictly for diagnostic analysis, not pass/fail scoring, separating final-answer quality from execution efficiency.
4. **Claims List:** A set of atomic, independently verifiable claims that combine to form a comprehensive response to the prompt, grounded exclusively in tool outputs. This claims based approach enables granular partial-credit scoring and error attribution.

This schema draws inspiration from programmatic verification frameworks [15] but adapts them for scalable, text-based final-answer evaluation while retaining rich diagnostics. An illustrative example task is provided in Appendix C.1.

Exposure and Distractor Policy. To model realistic agent contexts while preventing brute-force enumeration, the harness exposes a limited set of tools per task, augmented with distractors. Distractors are typically sampled from the same servers (e.g., `google-maps_maps_distance_matrix` vs. `google-maps_maps_geocode`; `desktop-commander_get_config` vs. `desktop-commander_read_file`) to ensure that success depends on accurate discovery and parameterization rather than simple server name recognition.

3.3 Environment Buckets and Taxonomy

We group servers into five environment buckets used for exposure analysis and domain-specific metrics. Table 2 lists representative servers and recurring pitfalls captured by our diagnostics.

To ensure balanced coverage, MCP-Atlas employs complementary strategies for task assignment:

- **Server-First Bucketing:** Tasks are generally open ended in terms of server selection, with authors freely selecting subsets of their choice. Throughout the data collection process, as server distributions form, tasks are assigned target servers from underrepresented categories. Authors are required to create prompts that naturally invoke at least one server from the list, enforcing targeted tool usage implicitly and providing a flexible approach to maintain balanced server representation.
- **Post-hoc LLM Categorization:** An LLM classifier assigns domain labels after creation for standardized reporting.

Server usage varies across the benchmark, with higher representation of general-purpose servers such as search and filesystem (see Appendix H for the full distribution).

3.4 Quality Assurance

We employ numerous quality checks throughout the data collection process to ensure results are robust and reputable. The following processes are used:

- **Human Review Layers:** Tasks undergo multiple rounds of human review. Initial task drafts are reviewed by domain experts for correctness, solvability, and adherence to authoring guidelines. A second review layer validates that prompts are appropriately natural (avoiding tool name leakage).
- **Auto Verification with LLMs:** Post-creation, tasks are automatically evaluated using LLM pipelines to verify that the claims list is complete and that the reference trajectory produces outputs consistent with the expected claims.
- **Quality Control Samples:** A random sample of tasks undergoes additional manual review to identify systematic issues and ensure consistent quality across the benchmark.

4. Evaluation Methodology and Experimental Setup

4.1 Evaluation Protocol

To ensure objective and reproducible assessment, we adopt a scoring system centered on the claims, supplemented by detailed diagnostics. This protocol balances the simplicity of primary scoring with the granularity required for in-depth error analysis, drawing from established practices in MCP evaluations [15–17].

Scoring via Claim-Based Rubric

The core evaluation metric is a pass/fail judgment based solely on the agent's final answer compared against the claim list. This decouples scoring from the specific execution path, rewarding the quality of the final outcome even if the agent's trajectory deviates from the reference one.

The scoring procedure operates as follows:

1. **Claim Verification:** Each claim in the claim list, defined as independently verifiable facts that the correct answer must contain, is checked against the agent's final answer. A claim is marked as *fulfilled* if the corresponding fact is present and correct in the response, *partially_fulfilled* if the corresponding fact is partially addressed in the response, or *not_fulfilled* if the corresponding fact is not present or incorrect in the response.
2. **Coverage Scoring:** The task's **coverage score** is computed as the average of each claim's coverage scores based on the following values: fulfilled = 1, partially fulfilled = 0.5, and not fulfilled = 0. Coverage $= \frac{|\text{sum claim scores}|}{|\text{total claim count}|}$.
3. **Pass/Fail Determination:** A task *passes* if the coverage score exceeds a predefined threshold; partial credit is recorded as the coverage fraction for analysis. We adopt a threshold of 0.75, which we selected as the point at which a response sufficiently addresses the prompt's core requirements. This allows tolerance for minor omissions while ensuring the majority of requested information is present and correct. To verify robustness, we conducted a sensitivity analysis by varying the threshold across 0.65, 0.75, and 0.85. Model rankings remain stable across this range, confirming that our findings are robust to the choice of threshold. (see Appendix E for details).

This claims-based approach ensures objectivity: scoring depends only on whether the agent's answer contains the correct factual information derived from tool outputs, not on stylistic factors like verbosity, formatting, or phrasing. Edge cases (e.g., responses involving tool retries or alternative phrasings that still convey the correct information) are clarified in Appendix B.

Judge Model and Human Alignment. Claims verification is performed using Gemini 2.5 Pro as the judge model. For each claim in the claims list, the judge determines whether the agent's response contains the required factual information, marking it as *fulfilled*, *partially_fulfilled*, or *not_fulfilled*. Unlike holistic LLM-as-judge approaches that rate overall response quality, our claims-based decomposition constrains the judge to factual verification grounded in tool outputs with known correct values.

To validate judge reliability, we conducted a human alignment study on a random sample of 100 tasks. Three human annotators independently rated each claim using the same rubric. The LLM judge achieved 78% agreement with the majority human vote, indicating substantial alignment. Given that our claims are designed to be independently verifiable facts with unambiguous ground truth, this agreement level supports the reliability of our automated evaluation.

4.2 Diagnostics

Beyond the pass/fail metric, we classify each failed task into one of four failure categories based on its dominant failure mode:

- **Tool Usage:** Failures related to incorrect tool selection, wrong parameters, schema violations, or improper sequencing of tool calls.
- **Task Understanding:** Failures due to premature stopping, missing subgoals, or incomplete interpretation of the task requirements.
- **Response Quality:** Failures where tool calls were correct but the final synthesis was inaccurate or contained hallucinated information.
- **Logical Errors:** Flawed reasoning about tool outputs or incorrect conditional logic in multi-step workflows.

Each failed task is assigned to exactly one dominant category by Gemini 2.5 Pro, which has access to the ground truth and the agent's generated trajectory. To validate reliability, we conducted a human alignment study: three annotators independently classified a random sample of 100 failed tasks, and the LLM judge achieved 78% agreement with the majority human vote. Disagreements primarily occurred in ambiguous cases where failures could plausibly span multiple categories (e.g., a missing tool call that also reflects incomplete task understanding). In Section 5.2, we analyze failure distributions across models.

4.3 Environment and Harness

The execution environment and harness are designed to ensure reproducibility, fairness, and fidelity to real MCP deployments. The design centers on (i) isolation of real MCP servers, (ii) controlled exposure of tools per task, and (iii) standardized logging for diagnostics.

1. **Real Servers** Tasks execute against containerized MCP servers with sandboxed filesystems and allow-listed egress, mirroring secure deployments. Containers are version-pinned and restarted per run to ensure a clean initial state.
2. **Per-Task, Controlled Tool Surface.** The harness exposes 10-25 tools per task, including 3-7 required tools and 5-10 distractors, to probe discovery and parameterization under noise. The harness mounts only the servers specified in the task creation, preventing off-policy tool access.
3. **Schema Enforcement.** All tool invocations must conform to server-declared schemas. The harness enforces JSON typing, required/optional parameter checks, and basic value validation.

4.4 Experimental Setup

Datasets and Splits

Experiments are conducted on the MCP-Atlas dataset (1,000 tasks). The dataset composition reflects the characteristics described in §3 (3-6 tool calls, the vast majority multi-server, one-third conditional branching). The distribution across environment buckets is approximately: BASIC (30-35%), ANALYTICS (10-15%), PRODUCTIVITY (20-25%), FINANCIAL (10-15%), and CODING (15-20%).

Models and Agents

We evaluate a representative set of frontier and open models, including Claude Opus 4.5, Claude Sonnet 4.5, o3, GPT-5, Grok-4, GPT-4o, Gemini 2.5 Pro, and Gemini 3 Pro (specific versions pinned in Appendix F). To isolate model capabilities, all models are driven by the same lightweight MCP client wrapper. This wrapper facilitates single-turn planning with multi-call execution, maintains tool schemas verbatim for schema-grounded function calling, and enforces execution budgets.

Execution and Scoring

All runs are executed in the containerized environment described in §4.3. For each model and split, we run the benchmark. Primary scoring utilizes the claims-based rubric detailed in §4.1. Diagnostics are not included in the public harness. We release container specifications, tool manifests, and harness configurations to ensure reproducibility (Appendix F).

5. Results and Analysis

We examine overall performance, diagnostic metrics that reveal specific failure patterns, and cross-domain variation. Throughout, we use pass rate as our primary metric, complemented by claims-based coverage scores that capture partial progress on complex tasks.

5.1 Overall Performance

Table 3 presents the pass rates and coverage scores across all evaluated models. While frontier models demonstrate meaningful competency on tool use, significant room for improvement remains. The best-performing model, Claude Opus 4.5, achieves a 62.30% pass rate, followed by Gemini 3 Pro at 54.10% and GPT-5 at 44.50%. These results demonstrate that end-to-end correctness on complex multi-step workflows remains a critical challenge.

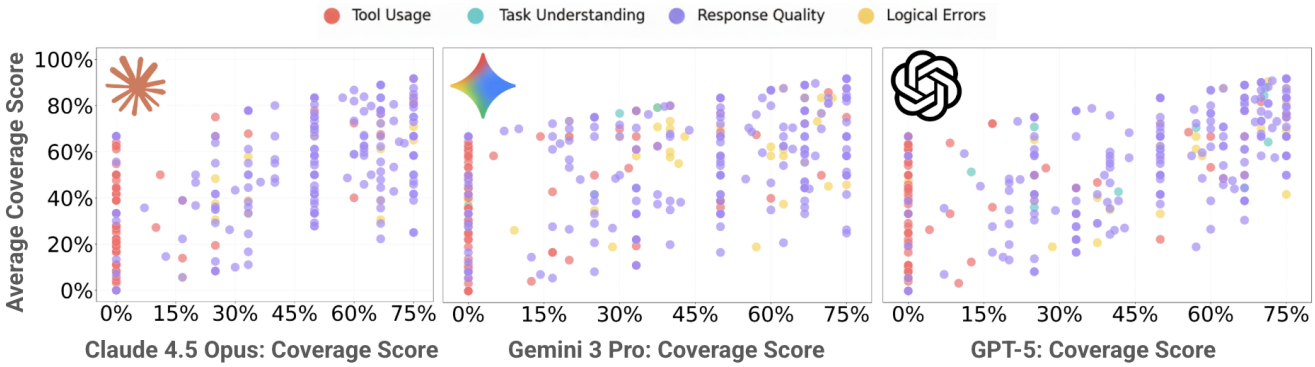
Table 3: MCP-Atlas pass rates and coverage scores across evaluated models.

Model	Pass Rate (%)†	Mean Coverage (%)
Claude Opus 4.5	62.3	78.5
Gemini 3 Pro	54.1	73.2
GPT-5	44.5	61.7
Claude Sonnet 4.5	43.8	62.1
o3 Pro (high)	43.6	66.9
Claude Opus 4.1	40.9	64.9
Claude Sonnet 4	35.6	57.3
GLM-4.5 Air	34.0	60.5
Kimi K2 Instruct	23.9	50.4
Qwen3-235B-A22B	12.0	29.0
Gemini 2.5 Pro	8.8	30.7
GPT-4o	7.2	28.5
Gemini 2.5 Flash	3.4	17.8

†Pass rate = percentage of tasks with coverage $\geq$75%.

Coverage measures how close a model's answers get to the ground truth across all tasks, including failures. We compute per-task coverage via claims-level scoring (0 / 0.5 / 1) and then average across tasks. As shown in Table 3, coverage and pass rate are tightly correlated–models with higher coverage generally achieve higher pass rates. The interquartile range (IQR) for coverage spans 28.8%–61.2%, indicating substantial variation in model capabilities. Claude Opus 4.5 achieves the highest average coverage at 78.5%, while lower-performing models achieve coverage below 20%. This gap underscores the wide disparity between frontier models and other alternatives on complex tool-use tasks.

Figure 2: Diagnostic categories for the top 3 models on failed tasks. The y-axis shows average coverage score across all models (i.e. task difficulty, higher average coverage score indicates lower task difficulty), while the x-axis shows model score. Colors indicate failure reasons, classified by an LLM judge with access to the ground truth and generated trajectory.



5.2 Failure Analysis

To understand why models fail, we categorize failures into four diagnostic categories: **Tool Usage**, **Task Understanding**, **Response Quality** and **Logical Errors** as described in Section 4.2. We present their distribution across models in Table 4 and Figure 2.

Table 4: Failure mode breakdown on MCP-Atlas. For each model, values indicate the distribution of primary failure types across its failed tasks. Each failed task is assigned to its dominant failure category, so percentages sum to 100%.

	Failure Distribution (% of Failed Tasks)			
Model	Tool Usage	Task Understanding	Response Quality	Logical Errors
Claude Opus 4.5	47.5	36.0	11.0	5.5
Gemini 3 Pro	49.0	35.0	10.5	5.5
GPT-5	51.0	34.0	10.0	5.0
Claude Sonnet 4.5	52.0	33.5	9.5	5.0
o3 Pro (high)	53.0	33.0	9.5	4.5
Claude Opus 4.1	54.5	32.0	9.0	4.5
Claude Sonnet 4	56.0	31.0	8.5	4.5
GLM-4.5 Air	57.5	30.0	8.5	4.0
Kimi K2 Instruct	59.0	29.0	8.0	4.0
Qwen3-235B-A22B	61.0	27.5	7.5	4.0
Gemini 2.5 Pro	63.0	26.0	7.0	4.0
GPT-4o	65.5	24.5	6.5	3.5
Gemini 2.5 Flash	68.5	22.5	5.5	3.5
Average	*56.7*	*30.3*	*8.5*	*4.5*

From these results, we observe the following:

Tool Use Errors Dominate. Errors related to TOOL USAGE are the most prevalent failure category across all models, ranging from 47.5% to 68.5% of failures (average: 56.7%). This highlights that correct tool selection and parameterization remain the primary challenges, with weaker models exhibiting significantly higher tool usage failure rates.

Task Understanding. The second major category is TASK UNDERSTANDING, accounting for 22.5–36.0% of failures (average: 30.3%). Models frequently stop prematurely or miss subgoals required for complete task

resolution. Notably, stronger models maintain higher task understanding scores, suggesting that improved reasoning capabilities translate to better multi-step planning.

Response Quality. Issues related to final response synthesis account for 5.5–11.0% of failures (average: 8.5%). These failures occur when tool calls were correct but the final answer was inaccurate or contained hallucinated information. Notably, as tool usage competency improves (as seen in frontier models), response quality becomes a proportionally larger contributor to failures.

Logical Errors. Flawed reasoning about tool outputs or incorrect conditional logic accounts for 3.5–5.5% of failures (average: 4.5%). This is the least common failure category, suggesting that when models correctly execute the necessary tools, they generally reason about the results effectively.

Table 5 details the most frequent specific failure modes and their mapping to the four diagnostic categories. Notably, "No tools called" (36.0% average) under TOOL USAGE and "Partial task completion" (25.8% average) under TASK UNDERSTANDING are the dominant failure patterns across all models. This mirrors the "unknown-tools" issues observed in related work [15], where agents fail to recognize that tools are available or stop prematurely before completing multi-step tasks.

Table 5: Granular breakdown of Failure modes

Diagnostic Category	Failure Mode	Avg. Share (%)
Tool Usage	No tools called	36.0
	Incorrect tool parameters	14.2
	Missing tool calls	6.5
Task Understanding	Partial task completion	25.8
	Skipped steps	3.0
	Misunderstood task	1.5
Response Quality	–	8.5
Logical Errors	Flawed reasoning	3.5
	Other	1.0

Example Failure. The following is an example failure task. The enabled tool set, reference trajectory, and claims list for the relevant task can be found in Appendix C.1.

- **Prompt:** "I'm researching papers on advertisement effectiveness and comparing it to our own online database advertising data. There's a 2024 paper by jane castleman that deals with ad control effectiveness, can you get me the abstract? I believe it mentions ad locality, for which I will also need to ask you for the date of our campaign with the biggest engagement rate, started during the 2015-2023 period, and its locality."
- **Trajectory:** arxiv_search_papers ("Jane Castleman ad control effectiveness ad locality 2024") -> find paper abstract, memory_search_nodes ("campaign engagement rate biggest highest 2015 2016 2017 2018 2019 2020 2021 2022 2023 locality") -> find relevant data, memory_read_graph (N/A) -> continue searching for relevant data
- **Response:** "I found the 2024 paper by Jane Castleman that you mentioned. Here's the abstract: [abridged for paper]. However, regarding your internal campaign data from 2015-2023, I searched the available knowledge graph but couldn't find any information about advertising campaigns, engagement rates, or locality data. The knowledge graph appears to contain only construction company information and related personnel data. You may need to access your advertising database directly or provide additional context about where this campaign data might be stored."

The agent successfully completed the first subtask (retrieving the paper abstract via `arxiv_search_papers`). However, for the second subtask, the agent searched the `memory` server, which returned no relevant results. Rather than exploring other tools in the enabled set, the agent concluded the data was unavailable. In fact, the campaign data was stored in the `notion` server, which was also exposed to the agent. This failure to try alternative tools after an unsuccessful search—despite having relevant options available—exemplifies incorrect tool selection, the most common TOOL USAGE failure mode.

5.3 Cross-Domain Analysis

Building on the failure category framework established above, we now analyze how performance and failure patterns vary across different domains reflecting the underlying MCP servers (Table 6). Performance varies significantly by domain, with pass rates differing by up to 17 percentage points within the same model across domains.

Table 6: Pass rates (%) by environment bucket. Performance varies across domains, with ANALYTICS tasks generally showing the highest success rates while FINANCIAL and CODING tasks prove most challenging.

Model	ENV-BASIC	ENV-ANALYTICS	ENV-PRODUCTIVITY	ENV-FINANCIAL	ENV-CODING	Overall
Claude Opus 4.5	68.4	71.5	64.2	58.2	55.8	62.30
Gemini 3 Pro	59.8	62.1	55.4	50.6	47.2	54.10
GPT-5	49.2	51.8	45.6	41.2	38.5	44.50
Claude Sonnet 4.5	48.5	50.9	44.8	40.5	37.8	43.80
o3 Pro (high)	48.2	50.6	44.5	40.2	37.5	43.60
Claude Opus 4.1	45.2	47.5	41.8	37.8	35.2	40.90
Claude Sonnet 4	39.4	41.2	36.2	32.8	30.5	35.60
GLM-4.5 Air	37.6	39.4	34.6	31.2	29.1	34.00
GPT-4o	8.2	8.8	7.4	5.8	4.2	7.20

Domain Characteristics. Tasks involving structured data operations (ANALYTICS) generally show the highest success rates across models (e.g., 71.5% for Claude Opus 4.5), potentially due to more uniform API schemas and less ambiguity in queries. In contrast, FINANCIAL and CODING domains remain highly challenging–Claude Opus 4.5 achieves 58.2% in Finance and 55.8% in Coding, while GPT-4o achieves only 5.8% and 4.2%, respectively. These domains often require precise parameterization (e.g., date formats, ticker symbols, specific query syntax) and effective disambiguation among semantically similar tools.

Connecting Domain Performance to Failure Categories. To understand *why* certain domains are more challenging, we analyze how the failure categories from Section 5.2 distribute across each domain. Table 7 presents this breakdown, revealing distinct failure signatures.

Table 7: Failure category distribution (%) by domain, averaged across all models. Each row sums to 100%, showing the relative contribution of each failure type within that domain. The overall average aligns with Table 4.

Domain	Tool Usage	Task Understanding	Response Quality	Logical Errors
ENV-BASIC	48.0	35.5	11.0	5.5
ENV-ANALYTICS	45.5	33.5	14.0	7.0
ENV-PRODUCTIVITY	55.0	31.5	9.0	4.5
ENV-FINANCIAL	64.0	26.0	6.5	3.5
ENV-CODING	71.0	21.0	5.0	3.0
Overall Average	*56.7*	*30.3*	*8.5*	*4.5*

Key Insights from Domain-Failure Analysis.

- **Tool Usage Failures Scale with Domain Difficulty.** The most challenging domains (FINANCIAL: 64.0%, CODING: 71.0%) exhibit substantially higher TOOL USAGE failure rates compared to easier domains (BASIC: 48.0%, ANALYTICS: 45.5%). This 23+ percentage point gap indicates that specialized domains introduce greater tool selection and parameterization complexity. Financial APIs often expose multiple endpoints for similar functions (e.g., real-time vs. historical quotes, different data providers), while coding tools require precise syntax specification.
- **Task Understanding Inversely Correlates with Tool Complexity.** Interestingly, TASK UNDERSTANDING failures are *higher* in easier domains (35.5% for BASIC) than in harder ones (21.0% for CODING). This suggests that in

simpler domains, models more often fail due to premature stopping or incomplete goal decomposition, whereas in complex domains, failures occur earlier in the pipeline–at tool selection–before task understanding becomes the bottleneck.

- **Analytics Domain Shows Unique Response Quality Challenges.** The ANALYTICS domain exhibits the highest RESPONSE QUALITY failure rate (14.0%) and LOGICAL ERRORS rate (7.0%). This reflects the domain's requirement for accurate numerical synthesis and multi-step calculations. Even when models correctly invoke analytics tools, they may misinterpret aggregated results, apply incorrect units, or make computational errors when combining outputs.
- **Productivity Tools Require Better Orchestration.** The PRODUCTIVITY domain shows elevated TOOL USAGE failures (55.0%) despite moderate overall difficulty. Manual inspection reveals this stems from orchestration challenges: productivity tasks often require coordinating across multiple services (e.g., calendar, email, document tools) with interdependent parameters, leading to cascading errors when early tool calls return unexpected formats.

Specific Failure Modes by Domain.

Table 8 presents the top failure modes for each domain, providing actionable insight into domain-specific challenges. The domain-failure analysis reveals that improving overall benchmark performance requires targeted interventions:

Table 8: Failure analysis by domain, showing the dominant challenges in each area. Percentages indicate share of failures within each domain.

Domain	Top Failure Sub-category	Other Failure Sub-categories
ENV-BASIC	Partial task completion (31.0%)	No tools called (29.0%)
ENV-ANALYTICS	No tools called (30.0%)	Incorrect conclusion (13.5%)
ENV-PRODUCTIVITY	Partial task completion (27.5%)	No tools called (24.0%)
ENV-FINANCIAL	No tools called (38.0%)	Incorrect tool parameters (24.5%)
ENV-CODING	No tools called (42.0%)	Incorrect tool parameters (27.5%)

1. **For Financial/Coding domains:** Focus on tool call correctness and parameter validation. The high rates of "Incorrect tool parameters" (24–28%) combined with "No tools called" (38–42%) suggest that models struggle both with tool awareness and precise specification in these technical domains. Better few-shot examples or retrieval-augmented schema lookup could yield significant gains.
2. **For Basic/Productivity domains:** Address premature stopping through improved planning. The prevalence of "Partial task completion" (27–31%) indicates models need better mechanisms for tracking subgoal progress and recognizing when tasks remain incomplete.
3. **For Analytics domains:** Improve numerical reasoning and output synthesis. The elevated RESPONSE QUALITY (14.0%) and LOGICAL ERRORS (7.0%) rates suggest that even successful tool orchestration does not guarantee correct final answers when quantitative reasoning is required.
4. **Across all domains:** The persistent "No tools called" failure mode (ranging from 24% to 42%) represents a fundamental tool awareness gap that affects all domains and warrants investigation into prompt engineering or fine-tuning approaches that increase tool invocation rates.

5.4 Discussion

The results from MCP-Atlas reveal both the capabilities and limitations of current LLMs for realistic tool use. While the top model achieves a 62.30% pass rate, most models struggle significantly with complex multi-step workflows. The preceding sections provide granular diagnostics; here we synthesize these findings into broader implications.

The Tool Awareness-Capability Gap. The failure analysis reveals a perhaps counterintuitive finding: the primary bottleneck is not models' *ability* to use tools correctly, but rather their *awareness* that tools should be used and their *persistence* in completing multi-step workflows. When models do engage with tools and reach the synthesis stage, they generally succeed. This suggests that current fine-tuning and prompting strategies may over-emphasize tool execution mechanics while under-emphasizing tool invocation triggers. Even frontier models exhibit substantial tool awareness deficits.

Developmental Trajectory Across Model Tiers. The failure distribution data reveals a clear stratification pattern: as models improve, failures shift from early-stage (tool selection) to mid-stage (orchestration) to late-stage (synthesis). Benchmark designers should anticipate this shift, ensuring that evaluation rubrics adequately capture downstream failure modes as tool selection competency improves.

Claims-Based Evaluation: Strengths and Trade-offs. Our pass-rate-centric scorer, based on claims coverage with a 75% threshold, targets end utility while remaining style-agnostic and scalable. It has advantages in:

- **Trajectory Independence:** Early missteps do not cause failure if the final answer is correct, decoupling success from adherence to a specific execution path.
- **Partial Credit:** The claims-based rubric rewards models that satisfy most requirements even if they miss edge cases.
- **Scalability:** Automated claims verification avoids the cost and variance of holistic LLM-as-a-judge scoring.

However, this approach does not penalize inefficiency or reward elegant solutions. Future iterations may incorporate efficiency constraints (call budgets, latency limits) to better reflect production deployment requirements.

6. Limitations and Broader Impact

Limitations. While MCP-Atlas advances the evaluation of MCP-enabled agents by utilizing real servers, complex workflows, and objective claims-based scoring, several limitations exist.

1. **Single-Turn Interaction.** Real-world deployments are inherently interactive, involving clarification, error recovery, and dynamic replanning. Our design focuses on single-turn interactions to isolate core tool-use competency. Consequently, behaviors such as strategic backtracking or complex recovery strategies are observable in traces but not explicitly scored for task success.
2. **Potential for Text-Form Bias in Claims Scoring.** Although we use a claims-based rubric rather than holistic LLM-as-a-judge scoring, decisions regarding text normalization and paraphrasing can still introduce edge cases (e.g., terse vs. verbose summaries). While this risk is mitigated compared to known LLM-judge style biases, it is not entirely eliminated. Future integration of programmatic checks (format/static/dynamic evaluators) is planned for select domains.
3. **Efficiency Not Enforced.** We log efficiency diagnostics (duplicate calls, retries) but do not enforce strict latency or call caps. Without these budgets, agents might "brute-force" discovery. Future tracks will incorporate latency/call constraints to reflect production environments.
4. **Real API Stability.** MCP-Atlas relies on real, production MCP servers which may experience API changes, rate limits, or availability issues over time. We version-pin server package versions, but benchmark results may vary if underlying services change. We acknowledge this trade-off between realism and reproducibility.
5. **Read-Only Task Scope.** To preserve server state integrity when using live production APIs, MCP-Atlas focuses exclusively on read-only operations (queries, searches, data retrieval). Tasks involving write operations, state mutations, or persistent side effects (e.g., creating records, sending messages, modifying files) are not included. While this design choice ensures reproducibility and avoids unintended consequences, it limits coverage of agentic workflows that require state-changing actions–an important capability for real-world deployments that we plan to address in future work through controlled synthetic environments.

Broader Impact. MCP-Atlas aims to serve as a practical and safety-conscious probe of tool-augmented LLM competency.

- **Research Community:** A standardized, real-server benchmark with a reproducible harness can accelerate apples-to-apples comparisons and detailed ablations across discovery, schema grounding, and error recovery. This complements existing programmatic and judge-based evaluation suites.
- **Industry Applications:** Detailed diagnostics on selection, typing, and errors help practitioners prioritize the development of necessary guardrails (e.g., parameter validation, retry mechanisms) before deployment.
- **Safety and Misuse.** Execution against real servers entails potential risks (e.g., quota consumption, data exfiltration). We mitigate these risks through containerization, sandboxed filesystems, and allow-listed egress, and we exclude tasks requiring sensitive or personal data.
- **Future Directions.** Expansion to multi-turn interactions and integration of programmatic evaluators for time-varying domains will further enhance the benchmark's rigor and applicability.

7. Conclusion

We introduced MCP-Atlas, a large-scale benchmark designed to evaluate the tool-use competency of LLMs using real MCP servers. Spanning 36 servers and 220 tools, with 1,000 tasks, MCP-Atlas assesses agents on complex, multi-step workflows elicited by single-turn, natural language prompts that avoid naming tools. We employ a claims-based rubric with partial credit for scoring, supplemented by internal diagnostics covering discovery, parameterization, error recovery, and efficiency.

Evaluation results of frontier models reveal that top models achieve pass rates exceeding 50%, while next-best models score in the 20-40% range, with predominant failures in tool usage and task understanding, indicating substantial headroom for improvement. Within the rapidly evolving MCP evaluation ecosystem, MCP-Atlas complements judge-based systems and programmatic frameworks by establishing an end-quality-with-diagnostics baseline on real servers at scale.

By releasing the schema, containerized harness, and benchmark dataset, we aim to facilitate reproducible and actionable evaluation. The persistent gap between planning, precise parameterization, and reliable task completion highlights critical research targets: enhancing tool discovery under distractors, improving schema-grounded argument validation, and developing robust recovery strategies. We view MCP-Atlas as a foundation for these directions, bridging the gap between rubric-based and programmatic evaluation paradigms toward the development of dependable MCP agents.

References

[1] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring Massive Multitask Language Understanding. In ICLR, 2021. arXiv:2009.03300.

[2] P. Liang et al. Holistic Evaluation of Language Models (HELM). arXiv:2211.09110, 2022.

[3] S. Zhou et al. WebArena: A Realistic Web Environment for Building Autonomous Agents. arXiv:2307.13854, 2023.

[4] E. Z. Liu, K. Guu, P. Pasupat, T. Shi, and P. Liang. Reinforcement Learning on Web Interfaces using Workflow-Guided Exploration. In ICLR, 2018. arXiv:1802.08802. (Introduces MiniWoB++).

[5] T. Xie et al. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. arXiv:2404.07972, 2024.

[6] Y. Chai et al. A3: Android Agent Arena for Mobile GUI Agents. arXiv:2501.01149, 2025.

[7] Y. Qin et al. ToolLLM: Facilitating Large Language Models to Master 16,464 Real-World APIs. arXiv:2307.16789, 2023. (Introduces ToolBench dataset).

[8] Z. Guo et al. StableToolBench: Towards Stable Large-Scale Benchmarking on Tool Learning of LLMs. In Findings of ACL, 2024. arXiv:2403.07714.

[9] S. G. Patil et al. The Berkeley Function-Calling Leaderboard (BFCL): From Benchmarks to Real-World Evaluation. OpenReview, 2024/2025. (Leaderboard and methodology).

[10] S. Yao, N. Shinn, P. Razavi, and K. Narasimhan. -bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. arXiv:2406.12045, 2024.

[11] C. E. Jimenez et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? arXiv:2310.06770, 2023.

[12] G. Mialon et al. GAIA: A Benchmark for General AI Assistants. arXiv: 2311.12983, 2023. (ICLR 2024 version available).

[13] Model Context Protocol (MCP) Specification. modelcontextprotocol.io/specification/2025-03-26, 2025.

[14] Anthropic. Introducing the Model Context Protocol. anthropic.com/news/model-context-protocol, Nov 2024.

[15] Z. Luo et al. MCP-Universe: Benchmarking Large Language Models with Real-World Model Context Protocol Servers. arXiv:2508.14704, 2025.

[16] Z. Wang et al. MCP-Bench: Benchmarking Tool-Using LLM Agents with Real MCP Servers and Fuzzy Prompts. arXiv:2508.20453, 2025.

[17] Z. Liu et al. Automatic MCP-based Deep Evaluation for AI Agent Models (MCPEval). arXiv:2507.12806, 2025.

[18] G. Mo et al. LiveMCPBench: Can Agents Navigate an Ocean of MCP Tools? arXiv:2508.01780, 2025.

[19] X. Gao et al. MCP-RADAR: A Multi-Dimensional Benchmark for Evaluating Tool Use Capabilities in LLMs. arXiv:2505.16700, 2025.

[20] M. Chen et al. Evaluating Large Language Models Trained on Code. arXiv:2107.03374, 2021. (Introduces HumanEval).

[21] Y. Li et al. Toolathlon: A Multi-Agent Benchmark for Tool-Assisted Long-Horizon Planning. arXiv:2505.xxxxx, 2025.

[22] X. Wu et al. MCPMark: Benchmarking LLM Agents on CRUD Operations with MCP Servers. arXiv:2506.xxxxx, 2025.

[23] Y. Zhao et al. MCPVerse: Expanding the Action Space for Agentic LLMs. arXiv:2507.xxxxx, 2025.

[24] L. Chen et al. MSC-Bench: A Curriculum for Multi-Server Coordination in MCP Agents. arXiv:2508.xxxxx, 2025.

[25] H. Zhang et al. MCPToolBench++: Large-Scale Multilingual MCP Server Evaluation. arXiv:2509.xxxxx, 2025.

[26] Y. Huang et al. MetaTool Benchmark for Large Language Models: Deciding Whether to Use Tools and Which to Use. Proceedings of ICLR, 2024.

[27] J. Ye et al. ToolEyes: Fine-Grained Evaluation for Tool Learning Capabilities of Large Language Models in Real-world Scenarios. arXiv preprint arXiv:2401.00741, 2024.

[28] A. Yehudai et al. Survey on Evaluation of LLM-based Agents. arXiv preprint arXiv:2503.16416, 2025.

A. Appendix A: Environment Buckets and Detailed Diagnostics

Bucket Shares and Target Mix. The distribution of tasks across the environment buckets is as follows: BASIC (32%), ANALYTICS (12%), PRODUCTIVITY (22%), FINANCIAL (12%), and CODING (22%). Representative servers and tools are detailed in Table 2. Server usage varies across the benchmark, with higher representation of general-purpose servers such as search and filesystem (see Appendix H for the full distribution).

Server Selection Rationale. The 36 MCP servers included in MCP-Atlas were selected to provide broad coverage across commonly used domains and use cases. Selection criteria included: (1) availability of stable, well-documented MCP implementations; (2) diversity across functional categories (search, analytics, productivity, finance, coding); (3) presence of multiple related tools per server to enable meaningful distractor selection; and

(4) applicability to realistic multi-step workflows. The goal was to create a representative sample of the MCP ecosystem that tests agents across a variety of practical scenarios.

Detailed Diagnostics Descriptions. Table 9 provides formal descriptions of the diagnostic metrics collected during evaluation.

Table 9: Detailed descriptions of diagnostic metrics.

Diagnostic	Description
Discovery Precision	Fraction of tool calls made to the set of required tools (T_{req}).
Discovery Recall	Fraction of required tools used at least once during the task execution.
Schema/Type Correctness	Fraction of calls resulting in valid, well-typed outputs (i.e., not errors).
Parameter Correctness	Schema-aware similarity measure between the arguments provided by the agent and valid arguments.
Error Rate	Fraction of total calls that return an error token (Err_t).
Recovery Rate	Fraction of errors that are immediately followed by a successful retry or alternative call.
Efficiency (Calls)	Total number of tool calls and the count of duplicate calls per task.

B. Appendix B: Scoring Policy and Edge Cases

We clarify the claims-based scoring policy for specific edge cases encountered during evaluation:

- **Duplicate Calls:** Responses involving duplicate or redundant tool calls will pass if the final answer satisfies the claims. However, the efficiency diagnostic will be logged as low.
- **Hallucinated Calls:** If the agent attempts to call a tool not present in T_{expose} (a hallucinated call), the task can still pass if the claims are satisfied (achieved using available tools). Hallucinated calls are logged as discovery errors.
- **Order Violations:** Deviations from the sequence specified in the reference trajectory (π^*) do not automatically result in failure. Order violations lead to failure only if they result in incomplete claims coverage.
- **Retries:** Successful retries following an initial error do not penalize the claims score, but they are tracked in the Error Recovery diagnostic.

Judge Model and Configuration. Claims verification is performed using Gemini 2.5 Pro as the judge model. For each claim in the claims list, the judge determines whether the agent's response contains the required factual information. The judge prompt instructs the model to assess each claim independently, marking it as fulfilled, partially fulfilled, or not fulfilled based on the presence and correctness of the corresponding information in the response.

C. Appendix C: Authoring Templates, Checklists, and Example Task

Prompt Template. Authors utilize templates emphasizing implicit tool requirements, e.g., "Achieve [complex goal] by synthesizing information from available resources." Prompts must avoid telegraphing tool names or specific parameters.

Claims Template. The claims list must enumerate distinct, independently verifiable facts grounded solely in the expected tool outputs.

Construction Checklists. To ensure benchmark quality, all tasks undergo a rigorous checklist:

1. **Time-invariance:** The correct answer must not change over time.
2. **No Name Leakage:** Prompts must not contain explicit server or tool names.

3. **Tool Dependency:** The task must be unsolvable using the LLM's parametric knowledge alone.
4. **Unambiguity:** There must be only one correct final answer.
5. **Complexity:** The task must require multiple tool calls (target 3-6) and ideally involve cross-server orchestration or conditional logic.

C.1 Example Task

To illustrate the task schema, consider the following example:

Prompt: "I'm researching papers on advertisement effectiveness and comparing it to our own online database advertising data. There's a 2024 paper by jane castleman that deals with ad control effectiveness, can you get me the abstract? I believe it mentions ad locality, for which I will also need to ask you for the date of our campaign with the biggest engagement rate, started during the 2015-2023 period, and its locality."

Enabled Tools:

- notion_API-post-search
- notion_API-post-database-query
- notion_API-retrieve-a-database*
- notion_API-retrieve-a-block*
- notion_API-retrieve-a-page*
- arxiv_search_papers
- fetch_fetch*
- memory_read_graph*
- memory_search_nodes*
- slack_channels_list*
- slack_conversations_history*
- slack_conversations_replies*
- slack_conversations_search_messages*
- whois_whois_domain*
- whois_whois_tld*

*(Tools marked with * are distractors)*

Reference Trajectory:

1. arxiv_search_papers ("jane castleman ad locality 2024") → paper abstract
2. notion_API-post-search ("advertising") → find relevant database
3. notion_API-post-database-query (database_id: "21b97551-844e-8068-b387-fe7a56b04348") → campaign date

Claims List:

1. "There's a 2024 paper by Jane Castleman with the title 'Why am I Still Seeing This: Measuring the Effectiveness Of Ad Controls and Explanations in AI-Mediated Ad Targeting Systems'."
2. "The abstract of the paper with title 'Why am I Still Seeing This: Measuring the Effectiveness Of Ad Controls and Explanations in AI-Mediated Ad Targeting Systems' is: 'Recently, Meta has shifted towards AI-mediated ad targeting mechanisms [... abridged for paper]'."
3. "There's a tie between three advertising campaigns with an engagement rate of 15%."
4. "The starting dates of the three winning advertising campaigns are: 2022-06-24, 2019-09-20 and 2017-09-09."
5. "The localities of the three winning advertisement campaigns are: 'National', 'International' and 'International'."

D. Appendix D: Extended Results

Per-Server Error Rates. We observe significant variation in syntax and type error rates across servers. Financial servers exhibit the highest error rates (up to 45%), often due to strict requirements for date formats and symbol scoping.

Error Recovery Statistics. The average recovery rate across all models and tasks is approximately 60%, indicating that agents are often able to correct initial mistakes within the allowed execution budget.

E. Appendix E: Threshold Sensitivity Analysis

To validate the robustness of our pass/fail scoring methodology, we conducted a sensitivity analysis examining how model rankings change under different coverage thresholds. We evaluated three threshold values: 0.65 (lenient), 0.75 (default), and 0.85 (strict).

Table 10 presents the pass rates for all evaluated models across the three threshold settings. While absolute pass rates naturally vary with threshold choice–higher thresholds yield lower pass rates–the relative ordering of models remains remarkably stable.

Table 10: Pass rates (%) across different coverage thresholds. Model rankings remain consistent across threshold choices, with Spearman rank correlations ≥ 0.98 between all threshold pairs.

Model	$\theta = 0.65$	$\theta = 0.75$	$\theta = 0.85$
Claude Opus 4.5	71.40	62.30	53.80
Gemini 3 Pro	63.50	54.10	45.20
GPT-5	54.20	44.50	35.80
Claude Sonnet 4.5	53.70	43.80	34.90
o3 Pro (high)	54.80	43.60	35.10
Claude Opus 4.1	51.30	40.90	32.40
Claude Sonnet 4	45.80	35.60	27.20
GLM-4.5 Air	46.20	34.00	24.80
Kimi K2 Instruct	34.60	23.90	15.70
Qwen3-235B-A22B	20.80	12.00	6.40
Gemini 2.5 Pro	17.50	8.80	4.20
GPT-4o	15.30	7.20	3.50
Gemini 2.5 Flash	9.80	3.40	1.20

Rank Stability Analysis. We computed Spearman rank correlation coefficients between model rankings at each threshold pair:

- $\theta = 0.65$ vs. $\theta = 0.75$: $\rho = 0.989$
- $\theta = 0.75$ vs. $\theta = 0.85$: $\rho = 0.993$
- $\theta = 0.65$ vs. $\theta = 0.85$: $\rho = 0.982$

All correlations exceed 0.98, indicating near-perfect rank preservation across the tested threshold range. The only rank changes observed involve models with overlapping confidence intervals at the default threshold (e.g., GPT-5 and Claude Sonnet 4.5), where small differences in pass rate do not constitute statistically significant performance gaps.

Implications. This analysis confirms that the choice of a 0.75 threshold does not artificially advantage or disadvantage particular models. The strong rank correlation across thresholds demonstrates that our benchmark reliably distinguishes model capabilities regardless of the specific pass/fail cutoff employed. Researchers may adopt alternative thresholds based on their evaluation needs with confidence that relative model comparisons will remain valid.

F. Appendix F: Reproducibility Checklist and Artifacts

To ensure the reproducibility of the results presented in this paper, we provide the following artifacts and information:

- **Harness Configuration:** Detailed configuration files for the evaluation harness, including Dockerfiles and specifications for all MCP servers used in the benchmark.
- **Server Allow-Lists:** Egress control lists specifying the allowed network access for each containerized server.

- **Model Specifications:** Pinned model endpoints, versions, and inference parameters used for every reported run.
- **Dataset:** The full dataset of 1,000 tasks, including prompts, tool set configurations, reference trajectories, and claims lists.
- **Release Process:** The benchmark data and harness code are released via a public GitHub repository.

G. Appendix G: Formalization

We formalize MCP-Atlas as a single-turn, multi-call agentic evaluation over real MCP servers with controlled tool exposure, utilizing a reference trajectory for analysis and claims-based scoring. Our notation follows the standard agent-server view adopted in recent MCP evaluations.

G.1 Entities and Notation

Servers and Tools. Let $\mathcal{S} = \{s_1, ..., s_K\}$ be the set of MCP servers. Server s_i exposes a finite set of typed tools $T_i = \{t_{i,1}, ..., t_{i,|T_i|}\}$ following the MCP interface. The universe of tools is $T = \bigcup_{i=1}^{K} T_i$. Tools are assumed to be disjoint across servers.

Task. A task is defined as a tuple

$$\tau = (G, C, T_{\text{expose}}, \pi^*, \mathcal{C}^*),$$

where: (a) G is the goal specification; (b) C is the natural-language context/constraints; (c) $T_{\text{expose}} \subseteq T$ is the allow-listed tool set presented to the agent; (d) π^* is the reference trajectory, a minimal dependency-covering sequence of tool calls used for diagnostics; and (e) $\mathcal{C}^* = \{c_1, ..., c_m\}$ is the claims list, comprising disjoint, independently verifiable propositions that the correct answer must contain.

Targets vs. Distractors. We define $T_{\text{req}} \subseteq T_{\text{expose}}$ as the target tools required by π^*, and $T_{\text{dist}} = T_{\text{expose}} \setminus T_{\text{req}}$ as the distractors.

G.2 Interaction Model and Traces

Tool Signatures. Each tool $t \in T$ defines a typed relation

$$t : \text{Args}_t \to \text{Out}_t \cup \text{Err}_t$$

where Args_t is a JSON-schema-typed domain. Outputs are either a well-typed value in Out_t or an error token $e \in \text{Err}_t$ (e.g., schema violation, rate limit).

Interaction Model. Given a single-turn prompt (G, C), the agent issues tool calls from T_{expose}. A call at step k is $u_k = \langle t_k, a_k \rangle$, and the environment returns $o_k \in \text{Out}_{t_k} \cup \text{Err}_{t_k}$. The execution trace is $x_{1:K} = ((u_1, o_1), ..., (u_K, o_K))$. The agent must finally emit a textual answer $\hat{A}$.

Planning View. This interaction induces a finite-horizon POMDP. Rewards are attached post-hoc via the claims-based scoring function.

G.3 Claims-Based Scoring

Answer-Centric Ground Truth. Let $\mathcal{Y}$ be the space of model answers. The benchmark defines a scoring functional

$$S : \mathcal{Y} \times 2^{\mathcal{C}^*} \to [0, 1],$$

which computes a rubricized correctness score by checking $\hat{A} \in \mathcal{Y}$ against the claims list $\mathcal{C}^*$. Pass/fail is determined solely by $S(\hat{A}, \mathcal{C}^*)$, independent of the trajectory π^*.

H. Appendix H: Leaderboard Server Distribution

Table 11 presents the distribution of MCP servers used in the public 500-task leaderboard subset. The "Count" column indicates the number of tasks in which each server appears as a required tool (note that tasks typically require multiple servers, so counts sum to more than 500).

Table 11: MCP server usage in the 1000-task leaderboard subset. Servers are sorted by usage count.

Server	Task Count
oxylabs	155
filesystem	133
exa	108
wikipedia	99
mongodb	98
airtable	96
mcp-code-executor	93
ddg-search	92
national-parks	85
brave-search	83
cli-mcp-server	78
alchemy	76
weather-data	76
notion	75
lara-translate	72
e2b-server	72
git	71
mcp-server-code-runner	70
osm-mcp-server	68
open-library	68
arxiv	68
twelvedata	67
github	67
met-museum	66
calculator	65
desktop-commander	58
google-maps	56
clinicaltrialsgov-mcp-server	51
whois	49
fetch	49
slack	48
google-workspace	40
memory	36
pubmed	30
weather	26
context7	4