SGVEF-LOOP: Coverage-Guided Progressive Topological Exploration and Fact-Grounded Metamorphic Evaluation for MCP Agents

Zenghao Liu, Yansong Zhang*
School of Information, Renmin University of China
{zenghaoliu, yszh}@ruc.edu.cn

Abstract

The rapid expansion of the Model Context Protocol (MCP) ecosystem introduces a combinatorially complex tool space, rendering existing frameworks inadequate for comprehensive agent evaluation. To address this problem, we propose **SGVEF-LOOP**, a coverage-guided framework for progressive topological exploration and fact-augmented metamorphic testing. SGVEF operates via a synergistic closed loop: it navigates sparse regions using *adaptive sampling*, synthesizes *oracle-free metamorphic pairs* grounded in static knowledge, enforces *dual-constraint validation* to ensure consistency and solvability, and leverages *execution feedback* to iteratively optimize exploration. Deploying this framework yields a high-fidelity benchmark achieving 100% node coverage and saturating 80.54% of the theoretical transition bound (estimated via Chao1). Evaluation of 8 diverse MCP Agents reveals capability stratification and exposes critical behavioral anomalies—such as *reasoning instability*—that conventional metrics fail to capture. Consequently, this work establishes a generalizable paradigm for scalable, rigorous agent evaluation in dynamic environments. Our code and dataset are available at https://github.com/zh6zh/SGVEF-LOOP.

1 Introduction

As the standard interface connecting AI Agents with external tools, Model Context Protocol (MCP) (Anthropic, 2024b) is driving the paradigm shift from text generation to autonomous problem-solving. In just one year (Anthropic, 2025c), the MCP ecosystem has expanded rapidly: it now hosts over 10,000 active public MCP servers spanning major providers (including AWS, Cloudflare, Google Cloud, and Microsoft Azure) and is integrated into platforms such as ChatGPT, Cursor, Gemini, Microsoft Copilot, and Visual Studio Code. However, this scale introduces critical evaluation challenges by exposing agents to a combinatorially complex and dynamic tool space. Since existing benchmarks struggle to accommodate this complexity, there is a pressing need for a framework capable of adaptively exploring unknown and complex tool transition spaces and conducting rigorous stress tests. Such a framework is essential to comprehensively assess the robustness and capabilities of MCP Agents as autonomous problem solvers in realistic environments.

Early studies (Gao et al., 2025; Wu et al., 2025; Yan et al., 2025; Luo et al., 2025; Yang et al., 2025) relied on manually predefined static task sets, which are labor-intensive and limited in scale, failing to accommodate the expansive MCP Server ecosystem. To enhance flexibility, several studies (Hoang et al., 2025; Li et al., 2024; Wang et al., 2024b) introduced rule-based structured generation paradigms. However, these rigid templates restrict the exploration of complex tool transitions, limiting their ability to simulate real-world variability. Recently, LLM-based generators (Liu et al., 2025; Wang et al., 2025; Huang et al., 2025) have emerged, but they often overlook two critical conditions in large-scale scenarios. First, *tool parameter solvability*, which requires that execution arguments be strictly derivable from the initial context or predecessor tool outputs. Second, *invocation stability*, which ensures consistent agent behavior under semantic noise—a prerequisite for scenarios demanding strict determinism, such as financial settlement.

Governed by the hard constraints of parameter solvability and the soft constraints of semantic logic, the space of valid tool combinations forms a sparse, deep implicit subspace rather than a Cartesian product. ❶ *How can we define and effectively explore this latent transition space?* Furthermore, the absence of a theoretical boundary for this valid subspace leaves the evaluation without a quantifi-

* Corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 26573–26599

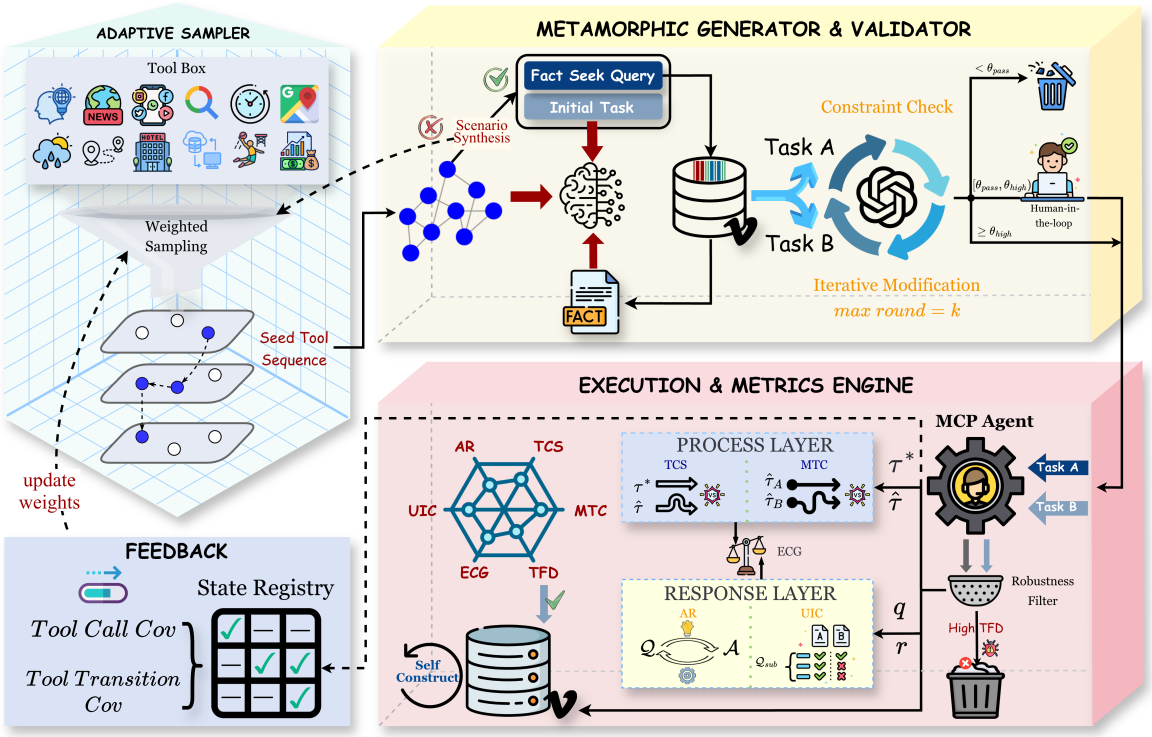


Figure 1: Overview of SGVEF-LOOP

able convergence target. Therefore, the core challenge arises: ❷ *Is there a mechanism to efficiently construct a large-scale, factually faithful benchmark that comprehensively covers this valid transition topology without prior manual mapping?* Addressing this is essential to accommodate the rapidly expanding tool ecosystem and achieve precise measurement of agent capabilities.

Drawing inspiration from coverage-guided fuzzing and metamorphic testing, we propose **SGVEF-LOOP** (Figure 1), a framework designed for progressive topological exploration and fact-augmented closed-loop verification. Departing from traditional random generation, SGVEF-LOOP models the latent tool transition space as a layered directed graph. It employs a state-feedback sampling mechanism, utilizing real-time feedback on tool invocation and transition coverage to drive agents toward maximal exploration of graph nodes and edges. To assess stability and overcome the Oracle Problem, we design a fact-augmented metamorphic testing generation pipeline. This pipeline features a collaborative verification mechanism—integrating LLM-as-a-Judge with Human-in-the-Loop—to rigorously filter out semantic inconsistencies and unsolvable parameters. Verified metamorphic pairs are then executed by a benchmark agent to validate robustness against weak semantic noise, with high-quality instances dynamically stored in a vector database as seeds for scalable expansion.

To quantify exploration completeness, we introduce the Chao1 estimator (Chao, 1984, 1987) to theoretically estimate the upper bound of the effective tool transition space. Large-scale experiments with foundation models demonstrate SGVEF-LOOP's rapid convergence: the framework achieves 100% tool invocation coverage by the 15th iteration and 80.54% transition coverage by the 60th, mining 1,080 effective tool transition paths. When utilizing this benchmark to evaluate 8 diverse foundation models, we observe distinct capability stratification and critical behavioral anomalies. Notably, our metamorphic analysis exposes pervasive reasoning instability under semantic noise, underscoring the urgent need for robust alignment in agentic systems.

To the best of our knowledge, this is the first work to propose a coverage-guided collaborative sampling mechanism tailored for MCP Agents. Through a progressive topological exploration strategy, we effectively overcome the evaluation complexities inherent in large-scale, dynamic tool spaces. Ultimately, this work establishes a general-

izable paradigm for generating comprehensive, robust, and high-fidelity metamorphic benchmarks.

2 Related Work

2.1 Evaluation for Tool-Use Agents

Existing studies on tool-use agent evaluation typically fall into two paradigms: static evaluation benchmarks and automated task generation.

Static Evaluation Benchmarks: This paradigm evaluates agent capabilities using predefined, fixed task sets. Early general benchmarks (Patil et al., 2025; Lu et al., 2025; Qin et al., 2023; Wang et al.; Liu et al., 2023; Zheng et al., 2023; Yao et al., 2024; Zhou et al., 2023) covered complex interaction scenarios but faced significant scalability limitations. First, they rely heavily on expensive manual curation, hindering rapid adaptation to evolving tools or novel scenarios. Second, they lack specific support for the MCP ecosystem. While recent studies such as MCP-Radar (Gao et al., 2025), MCPMark (Wu et al., 2025), MCPWorld (Yan et al., 2025), MCP-Universe (Luo et al., 2025), MCPSECBENCH (Yang et al., 2025) have shifted focus toward MCP-oriented evaluation, their reliance on static datasets inherently constrains scalability and coverage efficiency.

Automated Task Generation: To overcome the bottleneck of manual curation, research attention has shifted toward automated generation approaches. (1) Rule-Based Structured Generation. Studies such as (Hoang et al., 2025; Li et al., 2024; Wang et al., 2024b) extract facts from knowledge bases to construct tasks via templates or reasoning rules. However, the resulting restricted task space—often limited to Boolean queries or single-step invocations—fails to adequately simulate complex, unstructured real-world dynamics. (2) LLM as Generator. Alternatively, this approach utilizes seed tasks to bootstrap large-scale data synthesis. Early works (e.g., Wang et al., 2023; Li et al., 2023; Zhu et al., 2025; Qin et al., 2023) demonstrated the feasibility of this path. Recently, (Liu et al., 2025; Wang et al., 2025; Huang et al., 2025) have extended this paradigm to the MCP domain, incorporating task verification mechanisms. Notably, while MCPEval (Liu et al., 2025) achieves automation, it overlooks logical dependencies between tools. In contrast, MCP-Bench (Wang et al., 2025) and StateGen (Huang et al., 2025) explicitly model tool dependencies; StateGen further incorporates an energy sampling mechanism, enhancing the logical complexity and scenario diversity of the generated test cases.

Table 1: Comparison of SGVEF-LOOP with representative existing evaluation frameworks. **MCP-Focused**: Designed for MCP protocol; **Verif.**: Automated Task Validation; **Tool Logic**: Modeling tool dependencies; **Traj.**: Trajectory-based Evaluation; **Meta.**: Metamorphic Testing.
●: Supported; ○: Not supported.

Method	Scope & Capability			Methodology			
	MCP-Focused	Auto. Gen.	Verif.	Tool Logic	Traj.	Fuzzing	Meta.
Static Benchmarks							
BFCL (Patil et al., 2025)	○	○	○	○	●	○	○
AgentBench (Liu et al., 2023)	○	○	●	○	○	○	○
ToolBench 2.0 (Wang et al.)	○	○	○	●	○	○	○
MCP-Radar (Gao et al., 2025)	●	○	○	○	●	○	○
MCP-Universe (Luo et al., 2025)	●	○	○	○	○	○	○
MCPWorld (Yan et al., 2025)	●	○	○	○	○	○	○
Automated Frameworks							
ToolBench (Qin et al., 2023)	○	●	○	○	○	○	○
FuzzLLM (Yao et al., 2023)	○	●	●	○	○	●	○
API-Bank (Li et al., 2023)	○	●	●	●	●	○	○
Drowzee (Li et al., 2024)	○	●	●	○	○	○	●
MCPEval (Liu et al., 2025)	●	●	●	○	●	○	○
MCP-Bench (Wang et al., 2025)	●	●	●	●	●	○	○
StateGen (Huang et al., 2025)	○	●	●	●	●	●	○
SGVEF-LOOP (Ours)	●	●	●	●	●	●	●

2.2 Software Testing Methodologies for LLM Evaluation

Established software testing methodologies, notably fuzzing and metamorphic testing, have been widely adapted for LLM evaluation.

Fuzzing (Miller et al., 1990; Liang et al., 2018) is employed to enhance test case diversity. In safety alignment, various approaches aim to improve jailbreak detection, utilizing techniques such as AFL-based mutation (GPTFUZZER Yu et al., 2023), seed optimization (PROMPTFUZZ Yu et al., 2024), prompt decomposition (FuzzLLM Yao et al., 2023), and multi-turn backtracking (Crescendo Russinovich et al., 2024). Beyond safety, HFUZZER (Zhao et al., 2025) and MedFuzz (Ness et al., 2024) extend fuzzing to code hallucinations and medical robustness. However, existing work prioritizes textual over tool logic, leading to insufficient tool-call chain coverage and limiting the evaluation of agents' ability to handle complex, multi-step tasks.

To mitigate the Oracle Problem exacerbated by fuzzing-induced diversity, Metamorphic Testing (MT) (Chen et al., 2020) validates systems via logical relations. However, existing MT adaptations (Li et al., 2024; Eiras et al., 2025; Cho et al., 2025; Wang et al., 2024a) primarily verify semantic consistency under input perturbations, neglecting the critical validation of execution trajectories. A comprehensive feature comparison between our proposed framework and representative existing methodologies is summarized in Table 1.

3 Problem Formulation

We frame benchmark construction as a coverage-guided exploration process over a Latent Tool Transition Graph (LTTG), denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

Definition 1: Latent Tool Transition Graph (LTTG). Let $\mathcal{V} = \{t_1, \ldots, t_N\}$ be the observable finite set of atomic tools. The edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ represents valid transitions forming a sparse, implicit subspace governed by two constraints:

- Parameter Solvability ($\mathcal{C}_{solv}$): $\text{In}(t_j) \subseteq \mathcal{C}_0 \cup \bigcup_{t_i \prec t_j} \text{Out}(t_i)$, ensuring all required arguments for tool t_j are strictly derivable from the initial task context $\mathcal{C}_0$ or the outputs of predecessor tools.
- Semantic Coherence ($\mathcal{C}_{sem}$): Causal plausibility within real-world workflows.

Critically, the true topology $\mathcal{E}_{true}$ is unknown a priori, rendering the exploration space latent.

Definition 2: Metamorphic Test Unit. To enable oracle-free verification and assess execution robustness, we define a test unit as a metamorphic pair $\mathcal{M} = \{(q, \tau), (q', \tau)\}$. Here, q and q' are linguistic variants, sharing identical intent, grounded in an invariant trace $\tau = [v_1, \ldots, v_k]$ where $(v_i, v_{i+1}) \in \mathcal{E}_{true}$. Accordingly, the target benchmark is defined as a dataset $\mathcal{D} = \{\mathcal{M}_1, \ldots, \mathcal{M}_K\}$ composed of such validated units.

Definition 3: Graph Exploration Metrics. We introduce two hierarchical metrics to quantify the exploration of $\mathcal{G}$. Let $V(\tau)$ and $E(\tau)$ denote the sets of nodes and edges activated within trace τ:

- Node Coverage (C_{node}): Vertex diversity, defined as $C_{node} = |\bigcup_{\tau \in \mathcal{D}} V(\tau)|/|\mathcal{V}|$.
- Transition Coverage (C_{trans}): Topological completeness. Given the unknown cardinality $|\mathcal{E}_{true}|$, we employ the Chao1 Estimator (Chao, 1984, 1987) to approximate the effective boundary $\hat{S}_{chao1}$: $C_{trans} = |\bigcup_{\tau \in \mathcal{D}} E(\tau)|/\hat{S}_{chao1}$.

Optimization Objective. Ultimately, our goal is to synthesize a high-fidelity dataset $\mathcal{D}$ maximizing graph exploration under the topological constraints derived from $\mathcal{G}$

$$\begin{aligned} \max_{\mathcal{D}} \quad & \lambda_1 C_{node} + \lambda_2 C_{trans} \\ \text{s.t.} \quad & \forall \mathcal{M} \in \mathcal{D} : \mathcal{C}_{solv}(\mathcal{M}) \wedge \mathcal{C}_{sem}(\mathcal{M}) \end{aligned} \tag{1}$$

where λ_1 and λ_2 are weighting coefficients balancing node and transition exploration.

4 Methodology

Grounded in the LTTG formulation and the statistically derived boundary $\hat{S}_{chao1}$ (Appendix A), we propose SGVEF-LOOP. This closed-loop framework systematically navigates the latent transition space, synthesizing metamorphic test units to progressively approximate the theoretical limit.

Algorithm 1: SGVEF-LOOP Framework

Input: Tool Library $\mathcal{V}$, Agent $\mathcal{A}$, MaxIter K
Output: Benchmark $\mathcal{D}_{final}$

▷ Initialization
$\mathcal{D} \leftarrow \emptyset$, $\mathbf{V}_{obs} \leftarrow \mathbf{0}$, $\mathcal{E}_{obs} \leftarrow \emptyset$, $\hat{S} \leftarrow \infty$, $iter \leftarrow 0$
▷ Phase I: Topological Exploration Loop
while $iter < K$ ***and*** $|\mathcal{E}_{obs}| < \hat{S}$ **do**
  ▷ S: Adaptive Sampling
  if $C_{node} < 1$ **then**
    $S_{tools} \leftarrow \texttt{InvFreqWeighting}(\mathcal{V}, \mathbf{V}_{obs})$
  else $S_{tools} \leftarrow$ $\texttt{HierarchicalSample}(\mathcal{V}, \text{Category}, \text{Instance})$
  ▷ G: Fact-Grounded Generation
  $q_{init}, q_{seek} \leftarrow \texttt{ScenarioSynth}(S_{tools})$
  if $q_{init} == \emptyset$ **then continue** ▷ *Resample*
  $\mathcal{K} \leftarrow \texttt{StaticGrounding}(q_{seek})$
  $\mathcal{M} \leftarrow \texttt{MetaPairConstruct}(q_{init}, \mathcal{K})$
  ▷ V: Dual-Constraint Validation
  $S, \text{diag} \leftarrow \texttt{DualCritic}(\mathcal{M})$
  while $S < \theta_{pass}$ ***and*** *retry* **do**
    $\mathcal{M}, S \leftarrow \texttt{IterCorrect}(\mathcal{M}, \text{diag})$ ▷ Iterative Self-Correction
  end
  if $S < \theta_{high}$ ***and*** *(*$S < \theta_{pass}$ ***or*** *not* $\texttt{HumanReview}(\mathcal{M})$*)* **then continue**
  ▷ E & F: Exec & Feedback
  $\pi, \text{status} \leftarrow \texttt{BaseExec}(\mathcal{A}, \mathcal{M})$
  if *status* $==$ *RuntimeAnomaly* **then continue**
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathcal{M}\}$
  $\texttt{UpdateState}(\mathbf{V}_{obs}, \mathcal{E}_{obs}, \pi)$
  $\hat{S} \leftarrow \texttt{EstChao1}(\mathcal{E}_{obs})$
  $iter \leftarrow iter + 1$
end
▷ Phase II: Self-Constructed Evolution
$\mathcal{D}_{final} \leftarrow \texttt{VecDedup}(\texttt{SelfEvolve}(\mathcal{D}), \theta_{sim})$
return $\mathcal{D}_{final}$

4.1 S: Adaptive Sampling Strategy

To navigate the combinatorial transition space, simple random sampling over the theoretical Cartesian product is inefficient, as it predominantly yields invalid noise due to the sparsity of viable combinations. Therefore, we propose a Two-Phase Adaptive Sampling mechanism (Alg. 1, Lines 3-4). Acting as a necessary semantic filter to strictly confine sampling to valid transitions, this strategy dynamically shifts focus from atomic breadth to combinatorial depth.

① *Inverse-Frequency Weighting.* To mitigate long-tail distribution bias, we initially prioritize atomic coverage. We maintain a usage counter $\mathbf{V}_{obs}$ and compute sampling probabilities via an inverse-frequency Boltzmann distribution: $P(t_i) \propto$

$\exp(-\lambda \cdot \mathbf{V}_{obs}[i])$. This mechanism penalizes frequently sampled tools, driving exploration until full atomic coverage (C_{node}) is achieved.

② *Hierarchical Sampling.* Once C_{node} is satisfied, the objective shifts to transition coverage (C_{trans}). To counter the imbalance in server capacities, we employ a stratified strategy: (1) *Category Sampling*: Uniformly selecting $m \in [1, M_{max}]$ MCP servers to ensure balanced functional exposure; (2) *Instance Down-sampling*: Drawing a bounded subset of k tools within each selected category. This strategy constructs a balanced candidate set S_{tools}, facilitating the synthesis of complex, cross-domain scenarios.

4.2 G: Fact-Grounded Metamorphic Generation

Given S_{tools}, the synthesis process follows a cascaded three-stage pipeline (Alg. 1, Lines 5-8).

First, Scenario Synthesis assembles the sampled tools into a coherent context, yielding an Initial User Intent (q_{init}) and a Fact-Seeking Query (q_{seek}). A *Rejection Sampling* mechanism aborts functionally incompatible combinations, triggering an immediate fallback to the sampling phase.

Next, Static Knowledge Grounding utilizes q_{seek} to retrieve top-N relevant contexts ($\mathcal{K}$) from a Wikipedia vector database. Crucially, this retrieval is strictly limited to static attribute disambiguation to prevent hallucination.

Finally, Metamorphic Pair Construction integrates the intent q_{init} and the grounded facts $\mathcal{K}$ to synthesize a pair $\mathcal{M} = \{(q, \tau), (q', \tau)\}$, comprising two syntactically distinct queries that share identical semantic intent. This establishes a Metamorphic Relation (MR): semantic equivalence necessitates consistency in execution paths and final responses. This key property enables Oracle-free Evaluation, allowing the rigorous assessment of agent robustness against linguistic perturbations without relying on manual annotation.

4.3 V: Dual-Constraint Validation

To guarantee executability and logical soundness, we implement a Dual-Constraint Validation loop (Alg. 1, Lines 9-13).

An LLM-based Critic evaluates generated candidates against: ❶ *Semantic Consistency* ($\mathcal{C}_{sem}$, intent equivalence between q, q'), and ❷ *Parameter Solvability* ($\mathcal{C}_{solv}$, ensuring all arguments are strictly derivable from context or predecessor outputs). Unlike static filtering, rejection here triggers diagnostic feedback, guiding the generator to perform Iterative Self-Correction until constraints are met or the retry limit is reached.

To balance scalability with precision, we employ a Tiered Acceptance Strategy. High-confidence pairs ($S \geq \theta_{high}$) are automatically accepted, while marginal cases ($\theta_{pass} \leq S < \theta_{high}$) are routed to Human-in-the-Loop review. This hybrid mechanism ensures ambiguous scenarios are rigorously verified, preventing dataset contamination from subtle logic errors.

4.4 E & F: Execution and Feedback Loop

The final stage (Alg. 1 Lines 14-19) ensures the runtime validity and generates feedback signals.

Validated pairs are executed against a baseline execution agent $\mathcal{A}_{base}$. Unlike prior works prioritizing correctness (Liu et al., 2025), we implement a Robustness-Oriented Filtering strategy that decouples solvability from validity (Line 15). This preserves instances with high execution integrity regardless of the final prediction accuracy. This approach mitigates the *Benchmark Ceiling Effect* by retaining valid-but-unsolved cases, ensuring that the benchmark's complexity is not capped by the capabilities of the constructor model.

Post-execution, traces are analyzed to update the interaction counter $\mathbf{V}_{obs}$ and the transition registry $\mathcal{E}_{obs}$. This feedback signal propagates back to the Adaptive Sampler (Section 4.1), penalizing explored paths and steering the system toward under-explored topological regions.

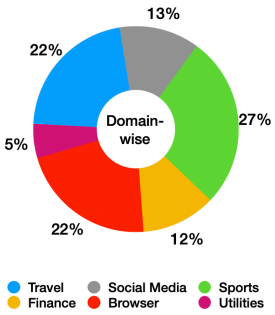
4.5 Self-Constructed Evolution

Post-convergence, the system transitions to Evolutionary Expansion (Phase II). Utilizing the high-fidelity dataset $\mathcal{D}$ accumulated in Phase I as seeds, we employ a *Self-Evolutionary Generation* process to scale benchmark volume. To mitigate redundancy, we enforce a Vector-based Semantic Deduplication constraint, retaining candidates only if their similarity to the existing corpus falls below θ_{sim}. This ensures $\mathcal{D}_{final}$ achieves substantial scale while maintaining semantic diversity.

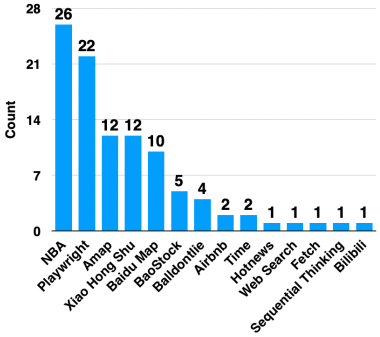
4.6 Evaluation Metrics

We employ a multi-dimensional evaluation protocol to comprehensively assess agent capabilities (formal definitions are provided in Appendix B).

(1) Process Fidelity & Robustness: We measure execution trace quality via *Tool Call Score (TCS)*



(a) Domain distribution of MCP tools.



(b) Tool count per MCP Server.

Figure 2: Distribution of the Tool Library.

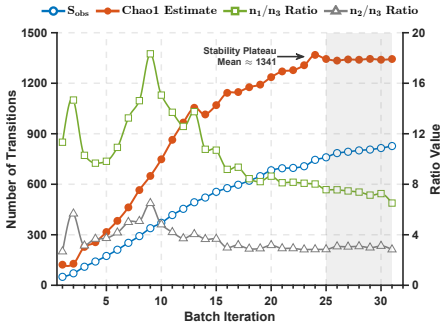


Figure 3: Analysis of transition space estimation. The shaded region (Batches 25-31) indicates the stability plateau.

and behavioral stability via *Metamorphic Consistency (MTC)*, while quantifying runtime reliability using *Trajectory Fault Density (TFD)*.

(2) Response Utility: Final output quality is evaluated through *Answer Relevance (AR)* (Es et al., 2024)—a hybrid of forward judgment and backward reconstruction—and *Unified Intent Coverage (UIC)* for atomic information completeness.

(3) Planning-Outcome Alignment: To diagnose discrepancies between tool usage and reasoning, we introduce the *Execution-Completion Gap (ECG)*, which quantifies the divergence between planning performance and final response utility.

5 Experiments

5.1 Research Questions (RQs)

We evaluate the effectiveness of the SGVEF-LOOP framework and the quality of the generated benchmark by answering the following RQs.

RQ1: How efficiently does SGVEF-LOOP navigate the sparse and constrained topology of the tool transition space?

RQ2: Can the constructed benchmark effectively differentiate the capabilities of diverse MCP Agents across multiple dimensions?

RQ3: What critical vulnerabilities and behavioral inconsistencies in agents are revealed by the metamorphic testing paradigm?

5.2 Experimental Setup

Tool Library. As illustrated in Figure 2, we compiled a tool library comprising 14 MCP Servers with 100 tools spanning 6 domains. Detailed configurations are provided in Appendix D.

Model Configuration. As detailed in Table 2, we selected a diverse suite of 8 representative LLMs, categorized by their roles as the *Framework Backbone* and the *Subject Agents*.

Table 2: Configuration of Large Language Models.

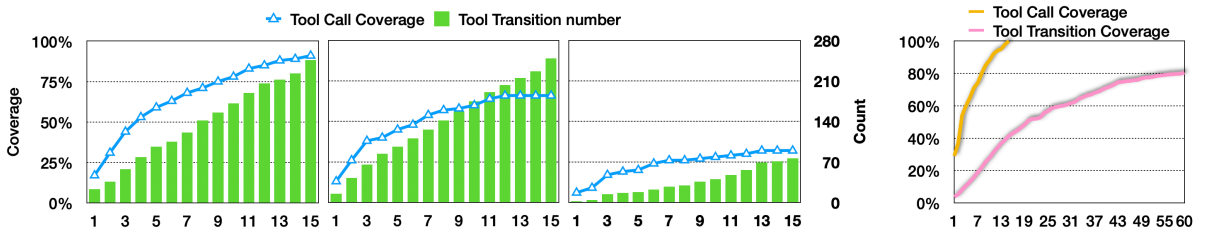
Role	Category	Model Instantiation
I. Framework Backbone (SGVEF-LOOP Driver)		
Gen. & Exec.	*Pilot Study (Efficiency)*	Qwen3-Max (Qwen, 2025)
	Benchmark Construction	Claude 3.5 Sonnet (Anthropic, 2024a)
Verification Evaluation	*Check & Refine*	GPT-4o (OpenAI, 2024b)
	Automatic Judge	Claude 4 Sonnet (Anthropic, 2025b)
II. Subject Agents (Evaluation Targets)		
Closed-Source		Claude Haiku 4.5 (Anthropic, 2025a), GPT-5 mini (OpenAI, 2025)
		Claude 3.5 Sonnet (Anthropic, 2024a), GPT-4o mini (OpenAI, 2024a)
Open-Source		DeepSeek-R1 (DeepSeek-AI, 2025), Qwen3-Max (Qwen, 2025)
		DeepSeek V3.2 (DeepSeek-AI, 2024), Kimi K2 (Kimi, 2025)

Metrics. For evaluation (RQ2), we employ the 6 multi-dimensional metrics defined in Section 4.6.

5.3 Effectiveness of SGVEF-LOOP (RQ1)

To accurately quantify exploration completeness within the sparse transition space, we employed the Chao1 estimator (Chao, 1984, 1987) to infer the theoretical upper bound of valid edges. Following a 31-batch exploration, the estimate achieved asymptotic stability at 1,341 (Figure 3). We adopt this converged value as the definitive upper bound for the effective denominator (statistical validation detailed in Appendix A).

Subsequently, we conducted a 15-round ablation study to evaluate the contribution of individual framework components (Figure 4 (a)). We compared the full SGVEF-LOOP framework against two ablated variants: SGVE (removing the Feedback mechanism) and GVE (removing both Feedback and Sampling mechanisms). Results demonstrate that the Sampling Mechanism is critical for initial acceleration, compared to the blind search of GVE (which stagnated at 32%), SGVE utilizes stratified space partitioning to efficiently reach 66% coverage (2.1× vs GVE). Furthermore, the Feedback Mechanism effectively resolves exploration bottlenecks. While SGVE hit a saturation plateau at 66% around round



(a) Validate Efficiency: Comparison of tool coverage and transition discovery across SGVEF, SGVE, and GVE variants.

(b) Long-term Benchmark Construction: The evolution of Tool Call coverage and Tool Transition Coverage using the full SGVEF framework.

Figure 4: Analysis of Exploration Dynamics: Component Efficiency Verification and Long-term Convergence Trajectory.

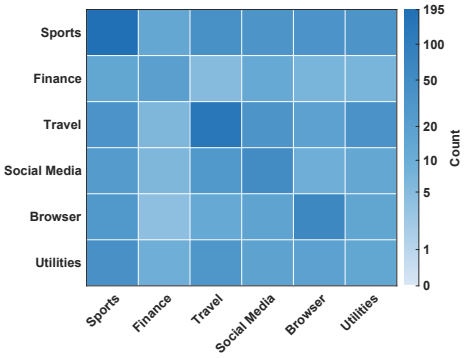


Figure 5: Tool Transition Domain-wise.

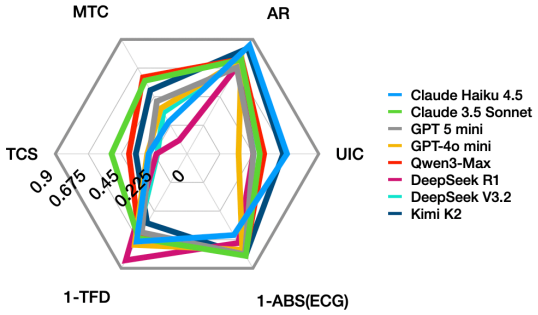


Figure 6: Performance trade-offs between Execution (Left: TCS, MTC) and Completion (Right: AR, UIC).

12, the full SGVEF-LOOP framework leveraged feedback-driven pruning to dynamically redirect search focus, sustaining momentum to achieve 91% coverage.

Building upon this validated efficiency, we deployed Claude 3.5 Sonnet as the backbone, extending exploration process to 60 rounds. As shown in Figure 4 (b), Tool Call Coverage (C_{node}) saturated rapidly, reaching 100% by round 15. Crucially, Tool Transition Coverage (C_{trans}) continued a steady growth trajectory well beyond this node saturation point. By round 60, SGVEF-LOOP identified 1,080 unique valid transitions. Compared to the estimated theoretical upper bound of 1,341, this yields a final C_{trans} rate of 80.54%, demonstrating that the framework can effectively map the majority of the feasible solution space within a reasonable computational budget.

To evaluate the structural diversity of the generated benchmark, we analyzed the transition distribution via a Domain-wise Transition Heatmap (Figure 5). The heatmap reveals high-density clusters along the diagonal blocks (e.g., Travel→Travel), indicating robust coverage of intra-domain tasks. More importantly, SGVEF-LOOP successfully activates off-diagonal regions, representing complex cross-domain explorations. Notably, valid interactions are observed even between semantically distinct domains, such as Finance↔Travel. This confirms that the benchmark captures a diverse spectrum of agent behaviors beyond simple single-domain scenarios.

> **Ans. to RQ1:** SGVEF-LOOP enables efficient navigation through synergistic mechanisms. Sampling accelerates initial discovery by 2.1×, while feedback breaks exploration plateaus. Thus, it saturates $\mathcal{C}_{node}$ (100%) by round 15 and maps 80.54% of the theoretical transition space by round 60.

5.4 Benchmarking MCP-based Agents(RQ2)

To address RQ2, we evaluated 8 MCP Agents on 576 valid metamorphic test cases (filtered from an initial set of 600). The results (Table 3, Figure 6) reveal distinct behavioral profiles:

Capability Stratification. The benchmark effectively differentiates agent planning capabilities, revealing a broad performance spread in Tool Call Score (0.203—0.516). Claude 3.5 Sonnet leads the cohort with a TCS of 0.516, demonstrating robust tool orchestration. Conversely, models such

Table 3: Comprehensive Evaluation of Diverse MCP Agents across Multiple Dimensions.

Model	Planning	Metamorphic Consistency					Response Quality		Execution Gap		Robustness
	TCS	MTC	*SA*	*SO*	*PA*	*FA*	AR	UIC	|ECG|	*Stdev*	TFD
Claude Haiku 4.5	0.263	0.251	0.243	0.313	0.077	0.302	**0.853**	**0.683**	**0.361**	0.197	0.310
Claude 3.5 Sonnet	**0.516**	0.575	0.564	**0.644**	0.372	0.643	0.742	0.497	0.198	0.152	0.337
GPT-5 mini	0.258	0.413	0.404	0.437	0.309	0.465	0.679	0.453	0.209	0.182	0.382
GPT-4o mini	0.272	0.358	0.400	0.373	0.263	0.416	0.741	0.352	0.245	0.147	0.284
Qwen3-Max	0.397	**0.602**	**0.592**	0.622	**0.445**	**0.688**	0.746	0.527	0.205	0.153	0.348
DeepSeek R1	0.209	0.107	0.107	0.122	0.041	0.132	0.691	0.453	0.299	0.149	0.163
DeepSeek V3.2	0.203	0.325	0.322	0.441	0.019	0.399	0.745	0.454	0.228	0.170	0.309
Kimi K2	0.350	0.501	0.500	0.576	0.212	0.600	0.830	0.657	0.201	0.144	**0.455**

Table 4: Case Study of Execution Divergence in Metamorphic Testing Pairs.

Category	Incomplete Planning	Poor Noise Robustness	Unstable Tool Selection
Task A	**Plan a comprehensive route** (driving, walking, public transit) from Beijing to Shanghai...	I work at **Beijing Financial Street**. Analyze Apple's stock...and check today's NBA scores.	...analyze popular **eco-friendly content** on **Xiaohongshu and Bilibili**...
Chain A	`transit integrated`, `driving`, `around search`, `walking`	`weather`, `today scoreboard`, `ticker info`, `ticker news`	`xhs search note`, `bilibili search`, `xhs auth status`
Task B	**Design a mixed route** (driving, walking, public transit) from Beijing to Shanghai...	I work at **Beijing Financial Street**. I need Apple's stock...and scores for today's NBA.	...understand **eco-friendly trends** on **Xiaohongshu and Bilibili**...
Chain B	`driving`, `text search`, `maps geo`	`today scoreboard`, `web search`, `ticker info`, `maps geo`, `ticker news`	`hotnews`, `xhs auth status`
Analysis	**Chain B missing tools.** Phrasing shift (comprehensive → mixed) caused `transit/walking drop`.	**Context hallucination.** User location (Financial St) triggered irrelevant `weather/geo tools`.	**Intent degradation.** Understand trends triggered generic `hotnews` instead of specific vertical search.

as DeepSeek V3.2 struggle (TCS ≈ 0.20). This substantial variance confirms that our generated tasks provide a rigorous stress test for real-world complexity.

Reasoning Stability. Qwen3-Max exhibits the highest behavioral consistency (MTC: 0.602), particularly excelling in Function Agreement (0.688). This indicates that despite potential variations in step-by-step execution, its underlying logical grounding remains robust under semantic perturbations. In contrast, DeepSeek R1 shows an extremely low Sequence Agreement (0.107), suggesting that reasoning-heavy models may over-optimize their planning, frequently generating divergent execution paths for identical intents.

Execution-Response Alignment. Claude Haiku 4.5 presents a notable anomaly: it achieves the highest AR (0.853) and UIC (0.683) alongside mediocre execution metrics (TCS: 0.263, MTC: 0.251), resulting in the most pronounced |ECG| (0.361). This disconnect implies a reliance on internal parametric knowledge to synthesize plausible answers rather than faithful tool execution. Conversely, Claude 3.5 Sonnet and Kimi K2 maintain minimal gaps ($|\text{ECG}| \approx 0.20$), indicating that their final responses are grounded in actual tool execution results.

Interaction Friction. Kimi K2 exhibits a high TFD (0.455) despite maintaining a decent TCS, reflecting a trial-and-error strategy characterized by frequent in-context error corrections. On the other end of the spectrum, DeepSeek R1 demonstrates the lowest TFD (0.163) coupled with a low TCS (0.209). This points to a conservative failure mode where the agent aborts execution early rather than attempting autonomous recovery.

> **Ans. to RQ2:** The benchmark reveals marked capability stratification across models and discerns distinct behavioral profiles. Crucially, it isolates and quantifies latent agent phenomena: *reasoning instability*, *parametric hallucination*, and *interaction friction*.

5.5 Insights from Metamorphic Testing(RQ3)

Metamorphic testing reveals the latent fragility of agents when subjected to semantic perturbations. As detailed in Table 4, we categorize these vulnerabilities into three distinct failure patterns.

First, *Incomplete Planning*. This occurs when synonymous rephrasing disrupts the agent's reasoning chain, resulting in the omission of critical sub-tasks. For instance, altering a routing directive from *comprehensive* to *mixed* causes the agent

to inexplicably drop required transit components. This reveals a dependency on rigid lexical triggers rather than a robust semantic comprehension of the underlying objective.

Second, *Poor Noise Robustness*. This pattern demonstrates the agent's susceptibility to contextual distractors. As shown in our case study, irrelevant background entities (e.g., a user's incidental geographical location) frequently provoke spurious tool invocations like weather or map queries. This indicates a tendency to prioritize surface-level entity association over core intent extraction.

Finally, *Unstable Tool Selection*. This manifests when abstract instructions induce functional degradation. Under such conditions, agents often default to generic, suboptimal tools rather than invoking precise, domain-specific solutions. Specifically, softening the instruction from *analyze* to *understand* triggers a downgrade from specialized vertical searches to generic retrieval.

Collectively, these findings indicate that current agents lack stable task representations, rendering their execution pathways highly vulnerable to superficial textual variations. To complement this analysis, Appendix C provides a comprehensive empirical taxonomy of observed tool execution failures.

> **Ans. to RQ3:** Under semantic perturbations, agents exhibit acute lexical sensitivity that directly disrupts tool orchestration. This brittleness—rooted in a lack of stable task representations—necessitates targeted Supervised Fine-Tuning to achieve robust, intent-driven tool execution.

6 Conclusion

We propose SGVEF-LOOP, a coverage-guided framework that captures 80.54% of the theoretical transition bound for MCP Agents. Comprehensive benchmarking across eight models reveals marked capability stratification and identifies critical behavioral anomalies, notably *reasoning instability* and *parametric hallucination*. Furthermore, metamorphic testing exposes their inherent susceptibility to semantic perturbations. Collectively, these findings underscore the fragility of current task representations, necessitating targeted Supervised Fine-Tuning for robust execution invariance.

Limitations

Theoretical Boundary Estimation. The Chao1 estimator relies on statistical assumptions that may underestimate the true transition space within heterogeneous tool ecosystems. The unexplored 19.46% likely comprises rare, deep multi-hop transitions requiring exponentially larger sampling budgets to discover.

Validation Mechanisms. Our semantic equivalence verification relies on an LLM-as-a-Judge approach, supplemented by limited Human-in-the-Loop review. Consequently, the framework inherits its potential false positives and negatives inherent to both evaluation paradigms.

Exploration Cost-Effectiveness. As the framework progresses into deeper transition layers, the cost-to-benefit ratio deteriorates significantly. Subsequent iterations demand increasing computational resources while yielding diminishing marginal coverage gains. Specifically, our experiments indicate that coverage increments drop from $\sim$37.7% (rounds 1–15) to $\sim$5.4% (rounds 45–60). This decay in sample efficiency renders the exhaustive exploration of the remaining tail transitions economically impractical.

Acknowledgments

We thank all reviewers for their insightful comments and suggestions. This work was partially supported by the National Key Research & Develop Plan (2023YFB4503600).

References

Airbnb mcp server. https://github.com/openbnb-org/mcp-server-airbnb. Accessed: 2025-12-25.

Amap mcp server. https://github.com/devsapp/amap-maps-mcp-server. Accessed: 2025-12-25.

Baidu map mcp server. https://github.com/baidu-maps/mcp. Accessed: 2025-12-25.

Balldontlie mcp server. https://github.com/mikechao/balldontlie-mcp. Accessed: 2025-12-25.

Bilibili mcp server. https://github.com/34892002/bilibili-mcp-js. Accessed: 2025-12-25.

Fetch mcp server. https://github.com/modelcontextprotocol/servers/tree/main/src/fetch. Accessed: 2025-12-25.

Hotnews mcp server. https://github.com/wopal-cn/mcp-hotnews-server/. Accessed: 2025-12-25.

Nba mcp server. https://github.com/labeveryday/nba_mcp_server. Accessed: 2025-12-25.

Playwright mcp server. https://github.com/microsoft/playwright-mcp. Accessed: 2025-12-25.

Sequential thinking mcp server. https://github.com/modelcontextprotocol/servers/tree/HEAD/src/sequentialthinking. Accessed: 2025-12-25.

Time mcp server. https://github.com/modelcontextprotocol/servers/tree/main/src/time. Accessed: 2025-12-25.

Web search mcp server. https://github.com/pskill9/web-search. Accessed: 2025-12-25.

Xhs mcp server. https://algovate.github.io/xhs-mcp/. Accessed: 2025-12-25.

Yahoo finance mcp server. https://github.com/narumiruna/yfinance-mcp. Accessed: 2025-12-25.

Anthropic. 2024a. Claude 3.5 sonnet. https://www.anthropic.com/news/claude-3-5-sonnet.

Anthropic. 2024b. Introducing the model context protocol. https://www.anthropic.com/news/model-context-protocol.

Anthropic. 2025a. Claude haiku 4.5. https://www.anthropic.com/claude/haiku.

Anthropic. 2025b. Claude sonnet 4. https://www.anthropic.com/news/claude-4.

Anthropic. 2025c. Donating the model context protocol and establishing the agentic ai foundation. https://www.anthropic.com/news/donating-the-model-context-protocol-and-establishing-of-the-agentic-ai-foundation.

Anne Chao. 1984. Nonparametric estimation of the number of classes in a population. *Scandinavian Journal of Statistics*, 11:265–270.

Anne Chao. 1987. Estimating the population size for capture-recapture data with unequal catchability. *Biometrics*, pages 783–791.

Tsong Yueh Chen, Shing Chi Cheung, and Siu ming Yiu. 2020. Metamorphic testing: A new approach for generating next test cases. *ArXiv*, abs/2002.12543.

Steven Cho, Stefano Ruberto, and Valerio Terragni. 2025. Metamorphic testing of large language models for natural language processing. *arXiv preprint arXiv:2511.02108*.

DeepSeek-AI. 2024. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Francisco Eiras, Eliott Zemour, Eric Lin, and Vaikkunth Mugunthan. 2025. Know thy judge: On the robustness meta-evaluation of llm safety judges. *arXiv preprint arXiv:2503.04474*.

Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. 2024. Ragas: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 150–158.

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

Thai Hoang, Kung-Hsiang Huang, Shirley Kokane, Jianguo Zhang, Zuxin Liu, Ming Zhu, Jake Grigsby, Tian Lan, Michael S. Ryoo, Chien-Sheng Wu, Shelby Heinecke, Huan Wang, Silvio Savarese, Caiming Xiong, and Juan Carlos Niebles. 2025. Lam simulator: Advancing data generation for large action model training via online exploration and trajectory feedback. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 12921–12934.

Yuheng Huang, Da Song, Zhenlan Ji, Shuai Wang, and Lei Ma. 2025. Evaluating llms on sequential api call through automated test generation. *arXiv preprint arXiv:2507.09481*.

Kimi. 2025. Kimi-k2. https://github.com/kimi-k2/Kimi-K2.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*.

Ningke Li, Yuekang Li, Yi Liu, Ling Shi, Kailong Wang, and Haoyu Wang. 2024. Drowzee: Metamorphic testing for fact-conflicting hallucination detection in large language models. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA2):1843–1872.

Hongliang Liang, Xiaoxiao Pei, Xiaodong Jia, Wuwei Shen, and Jian Zhang. 2018. Fuzzing: State of the art. *IEEE Transactions on Reliability*, 67(3):1199–1218.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Yuxian Gu, Han Ding, Kai Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Shengqi Shen, Tianjun Zhang, Sheng Shen, and 5 others. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Zhiwei Liu, Jielin Qiu, Shiyu Wang, Jianguo Zhang, Zuxin Liu, Roshan Ram, Haolin Chen, Weiran Yao, Huan Wang, Shelby Heinecke, Silvio Savarese, and

Caiming Xiong. 2025. Mcpeval: Automatic mcp-based deep evaluation for ai agent models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 373–402.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. 2025. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1160–1183.

Ziyang Luo, Zhiqi Shen, Wenzhuo Yang, Zirui Zhao, Prathyusha Jwalapuram, Amrita Saha, Doyen Sahoo, Silvio Savarese, Caiming Xiong, and Junnan Li. 2025. Mcp-universe: Benchmarking large language models with real-world model context protocol servers. *arXiv preprint arXiv:2508.14704*.

Barton P Miller, Lars Fredriksen, and Bryan So. 1990. An empirical study of the reliability of unix utilities. *Communications of the ACM*, 33(12):32–44.

Robert Osazuwa Ness, Katie Matton, Hayden S. Helm, Sheng Zhang, Junaid Bajwa, Carey E. Priebe, and Eric Horvitz. 2024. Medfuzz: Exploring the robustness of large language models in medical question answering. *ArXiv*, abs/2406.06573.

OpenAI. 2024a. Gpt-4o mini: Advancing cost-effective artificial intelligence. https://openai.com/zh-Hant-HK/index/gpt-4o-mini-advancing-cost-efficient-intelligence/.

OpenAI. 2024b. Hello gpt-4o. https://openai.com/index/hello-gpt-4o/.

OpenAI. 2025. Gpt-5 debuts. https://openai.com/zh-Hant-HK/gpt-5/.

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

Yujia Qin, Shi Liang, Yining Ye, Kunlun Zhu, Lan Yan, Ya-Ting Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Marc H. Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Qwen. 2025. Qwen3-max. https://qwen.ai/blog?id=qwen3-max.

Mark Russinovich, Ahmed Salem, and Ronen Eldan. 2024. Great, now write an article about that: The crescendo multi-turn llm jailbreak attack. *ArXiv*, abs/2404.01833.

Jon Saad-Falcon, Omar Khattab, Christopher Potts, and Matei Zaharia. 2024. Ares: An automated evaluation framework for retrieval-augmented generation systems. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 338–354.

Guangyu Wang, Jianhong Liu, Meilin Zhou, Xiaoming Chen, Lihua Zhang, and Zhihao Sun. Toolbench 2.0: Evaluating long-horizon and multi-step tool use in llms.

Guanyu Wang, Yuekang Li, Yi Liu, Gelei Deng, Tianlin Li, Guosheng Xu, Yang Liu, Haoyu Wang, and Kailong Wang. 2024a. Metmap: Metamorphic testing for detecting false vector matching problems in llm augmented generation. *2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering (Forge) Conference Acronym:*, pages 12–23.

Wenxuan Wang, Juluan Shi, Zhaopeng Tu, Youliang Yuan, Jen-tse Huang, Wenxiang Jiao, and Michael R Lyu. 2024b. The earth is flat? unveiling factual errors in large language models. *arXiv preprint arXiv:2401.00761*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 13484–13508.

Zhenting Wang, Qi Chang, Hemani Patel, Sanjana Biju, Chen Wu, Quan Liu, Aolin Ding, Alireza Rezazadeh, Ankit Shah, Yujia Bao, and Eugene Siow. 2025. Mcp-bench: Benchmarking tool-using llm agents with complex real-world tasks via mcp servers. *arXiv preprint arXiv:2508.20453*.

Zijian Wu, Xiangyan Liu, Xinyuan Zhang, Lingjun Chen, Fanqing Meng, Lingxiao Du, Yiran Zhao, Fan Zhang, Yaoqi Ye, Jiawei Wang, Zirui Wang, Jinjie Ni, Yufan Yang, Arvin Xu, and Michael Shieh. 2025. Mcpmark: A benchmark for stress-testing realistic and comprehensive mcp use. *arXiv preprint arXiv:2509.24002*.

Yun-Yu Yan, Shihe Wang, Jiajun Du, Yexuan Yang, Yuxuan Shan, Qichen Qiu, Xianqing Jia, Xing Wang, Xin Yuan, Xu Han, Maosheng Qin, Yinxiao Chen, Chen Peng, Shangguang Wang, and Mengwei Xu. 2025. Mcpworld: A unified benchmarking testbed for api, gui, and hybrid computer use agents. *arXiv preprint arXiv:2506.07672*.

Yixuan Yang, Daoyuan Wu, and Yufan Chen. 2025. Mcpsecbench: A systematic security benchmark and playground for testing model context protocols. *arXiv preprint arXiv:2508.13220*.

Dongyu Yao, Jianshu Zhang, Ian G. Harris, and Marcel Carlsson. 2023. Fuzzllm: A novel and universal fuzzing framework for proactively discovering jailbreak vulnerabilities in large language models. *ArXiv*, abs/2309.05274.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *ArXiv*, abs/2406.12045.

Jiahao Yu, Xingwei Lin, Zheng Yu, and Xinyu Xing. 2023. Gptfuzzer: Red teaming large language models with auto-generated jailbreak prompts. *arXiv preprint arXiv:2309.10253*.

Jiahao Yu, Yangguang Shao, Hanwen Miao, Junzheng Shi, and Xinyu Xing. 2024. Promptfuzz: Harnessing fuzzing techniques for robust testing of prompt injection in llms. *ArXiv*, abs/2409.14729.

Yukai Zhao, Menghan Wu, Xing Hu, and Xin Xia. 2025. Hfuzzer: Testing large language models for package hallucinations via phrase-based fuzzing. *ArXiv*, abs/2509.23835.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Haotong Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyin Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. 2025. Multiagentbench: Evaluating the collaboration and competition of llm agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8580–8622.

A Estimation of the Effective Tool Transition Space via Chao1

A.1 Rationale for Statistical Estimation

Given our experimental environment comprising 100 distinct tools, the theoretical upper bound of pairwise tool transitions is defined by the Cartesian product. However, adopting this combinatorial maximum as the denominator for coverage analysis is methodologically unsound. In real-world semantic spaces, valid tool transitions are strictly constrained by two primary constraints:

(1) Parameter Solvability ($\mathcal{C}_{solv}$): A transition is valid only if the successor tool's required inputs can be strictly satisfied by either the given task context ($\mathcal{C}_0$) or the execution outputs of the predecessor tool, effectively forming a strict I/O constraint.
(2) Semantic Coherence ($\mathcal{C}_{sem}$): A transition must constitute a logically sound and purposeful step within a broader, goal-oriented execution plan, maintaining causal plausibility within real-world workflows.

Consequently, the effective tool transition space is a strict subset of the theoretical combinatorial space. Because the exact boundary of this effective space remains unknown a priori, we employ the Chao1 estimator (Chao, 1984, 1987) to statistically infer its true scale. Chao1 is a non-parametric method primarily designed to estimate the unknown total number of classes in a population. Rather than relying on high-frequency data, it extracts critical information from low-order occupancy numbers (i.e., the total number of observed classes, singletons, and doubletons). This method eschews strong assumptions such as a pre-specified class distribution or equal capture probabilities. Even under sampling heterogeneity introduced by adaptive sampling, the Chao1 estimate remains mathematically valid as a conservative lower bound of the actual population size.

The estimator is formally defined as:

$$S_{\text{chao1}} = S_{\text{obs}} + \frac{n_1^2}{2n_2} \tag{2}$$

Where: S_{obs} denotes the number of unique transitions observed; n_1 denotes the number of transitions observed exactly once (singletons); n_2 denotes the number of transitions observed exactly twice (doubletons).

A.2 Validation of Estimator

We executed the semantic exploration over 31 sequential batches using the SGVEF-LOOP framework, employing Claude 3.5 Sonnet as the backbone model. To ensure the statistical validity of the Chao1 estimator within our specific context, we rigorously verified three critical conditions throughout the process. The convergence trends and validation metrics are illustrated in Figure 3, with the supporting experimental data detailed in Table 5.

Condition 1: Computational Validity ($n_1 > 0, n_2 > 0$)**.** The Chao1 formula inherently relies

Table 5: Statistical validation data for Chao1 estimation. All batches are shown to demonstrate the transition from the exploration phase to the saturation phase. Note: n_1, n_2, and n_3 denote the counts of singletons, doubletons, and tripletons.

Batch	Key Metrics		Frequency Counts			Validation Ratios	
	S_{obs}	Chao1 Est.	n_1	n_2	n_3	n_1/n_3	n_2/n_3
1	50	122.25	34	8	3	11.33	2.67
2	71	127.94	44	17	3	14.67	5.67
3	110	227.82	72	22	7	10.29	3.14
4	141	255.68	87	33	9	9.67	3.67
5	174	316.24	108	41	11	9.82	3.73
6	211	382.61	131	50	12	10.92	4.17
7	252	462.68	159	60	12	13.25	5.00
8	292	565.48	190	66	13	14.62	5.08
9	339	649.26	220	78	12	18.33	6.50
10	371	748.15	241	77	16	15.06	4.81
11	417	863.88	274	84	20	13.70	4.20
12	454	966.38	302	89	24	12.58	3.71
13	493	1054.34	330	97	24	13.75	4.04
14	521	1014.61	334	113	31	10.77	3.65
15	555	1069.91	353	121	33	10.70	3.67
16	577	1142.92	367	119	40	9.18	2.98
17	597	1147.69	374	127	40	9.35	3.17
18	621	1176.38	380	130	45	8.44	2.89
19	648	1191.78	386	137	47	8.21	2.91
20	683	1236.14	406	149	47	8.64	3.17
21	695	1270.15	414	149	51	8.12	2.92
22	697	1277.72	416	149	51	8.16	2.92
23	707	1307.00	420	147	52	8.08	2.83
24	745	1368.34	441	156	55	8.02	2.84
25	760	**1344.00**	439	165	58	7.57	2.84
26	785	**1334.49**	446	181	59	7.56	3.07
27	793	**1341.37**	448	183	60	7.47	3.05
28	801	**1339.56**	450	188	61	7.38	3.08
29	806	**1344.56**	450	188	63	7.14	2.98
30	815	**1339.61**	450	193	62	7.26	3.11
31	827	**1343.93**	449	195	69	6.51	2.83

on rare observations to infer the unobserved population size. As shown in Table 5, both singleton (n_1) and doubleton (n_2) counts maintained strictly positive values from the initial batch onward, satisfying the fundamental computational prerequisites of the estimator.

Condition 2: Low-Order Information Dominance ($n_1, n_2 \gg n_3$). The robustness of the Chao1 estimator depends on the dominance of rare items in the frequency distribution. We monitored the ratios ($n_1/n_3, n_2/n_3$) to verify the distribution hierarchy. As illustrated in Figure 3, these ratios initially exhibit elevated values during the early exploration phase and gradually descend to a stable equilibrium. This trajectory confirms that, even as the empirical distribution matures, low-frequency counts consistently dominate high-frequency counts, thereby upholding the estimator's rare-item assumption.

Condition 3: Asymptotic Stability (Convergence). The most definitive evidence of the estimator's reliability is the convergence of its predicted value. The estimated trajectory in Figure 3 demonstrates two distinct phases:

- **Discovery Phase (Batches 1-24):** The Chao1 estimate rises steeply, reflecting the framework's rapid discovery of new elements within the semantic space.
- **Saturation Plateau (Batches 25-31):** Highlighted by the shaded region in Figure 3, a structural plateau emerges after Batch 25. While the observed count continues to grow ($760 \rightarrow 827$), the Chao1 estimate flattens significantly, oscillating around a mean of 1,341.

Based on the robust asymptotic behavior observed during this saturation, we adopt the converged mean of 1,341 as the definitive estimate for the total effective size of the tool transition space.

B Evaluation Metrics

We introduce a multi-dimensional evaluation framework to assess execution process fidelity,

metamorphic consistency, and response quality.

B.1 Process Fidelity & Robustness

Tool Call Score (TCS). To quantify the structural fidelity of the generated tool chain relative to the ground truth, we model the execution trace as an ordered sequence. Let τ^* and $\hat{\tau}$ denote the expected and actual tool sequences, respectively. We define TCS as a weighted aggregation of Sequence Recall (SR) and Sequence Precision (SP), computed via the Longest Common Subsequence:

$$\text{TCS} = \alpha \underbrace{\frac{|\text{LCS}(\tau^*, \hat{\tau})|}{|\tau^*|}}_{\text{SR}} + (1-\alpha) \underbrace{\frac{|\text{LCS}(\tau^*, \hat{\tau})|}{|\hat{\tau}|}}_{\text{SP}} \quad (3)$$

Metamorphic Tool Chain Consistency (MTC). To assess behavioral stability, we evaluate metamorphic pairs of traces $(\hat{\tau}_A, \hat{\tau}_B)$ derived from semantically equivalent queries. MTC is formulated as a weighted ensemble of four sub-metrics:

$$\text{MTC} = w_1\text{SA} + w_2\text{SO} + w_3\text{PA} + w_4\text{FA} \quad (4)$$

Let $L_{max} = \max(|\hat{\tau}_A|, |\hat{\tau}_B|)$. The sub-metrics are defined as follows:

- Sequence Agreement (SA): The normalized LCS length, defined as $|\text{LCS}(\hat{\tau}_A, \hat{\tau}_B)|/L_{max}$.
- Set Overlap (SO): The Jaccard similarity of the unique tool sets U, defined as $|U_A \cap U_B|/|U_A \cup U_B|$.
- Prefix Adequacy (PA): The ratio of the Longest Common Prefix length, $|\text{LCP}(\hat{\tau}_A, \hat{\tau}_B)|/L_{max}$.
- Function Agreement (FA): The normalized LCS computed at the functional category level $\mathcal{F}(\cdot)$, defined as $|\text{LCS}(\mathcal{F}(\hat{\tau}_A), \mathcal{F}(\hat{\tau}_B))|/L_{max}$, which rewards functionally equivalent invocations.

Trajectory Fault Density (TFD). To measure operational robustness, we calculate the frequency of runtime errors (e.g., parameter mismatches, timeouts) normalized by the total trajectory length:

$$\text{TFD} = \frac{1}{|\hat{\tau}|} \sum_{k=1}^{|\hat{\tau}|} \mathbb{I}(a_k \in \mathcal{E}_{fail}) \quad (5)$$

where a_k denotes the k-th action and $\mathbb{I}(\cdot)$ is the indicator function for the error state set $\mathcal{E}_{fail}$.

B.2 Response Quality

Answer Relevance (AR). We employ a hybrid metric combining forward LLM-based evaluation and backward semantic reconstruction to assess if the response r faithfully addresses the query q:

$$\text{AR} = \lambda \mathcal{J}(q, r) + \frac{1-\lambda}{N} \sum_{i=1}^{N} \cos(\mathbf{e}(q), \mathbf{e}(\hat{q}_i)) \quad (6)$$

where $\mathcal{J}$ denotes the score from an LLM Judge(Saad-Falcon et al., 2024), and $\hat{q}_i$ represents queries inversely generated from r (Es et al., 2024).

Unified Intent Coverage (UIC). To measure information completeness, we decompose the complex query into a set of atomic sub-questions $\mathcal{Q}_{sub}$ and evaluate the granular coverage:

$$\text{UIC} = \frac{1}{|\mathcal{Q}_{sub}|} \sum_{q_j \in \mathcal{Q}_{sub}} \text{Score}_{\text{model}}(r, q_j) \quad (7)$$

where $\text{Score}_{\text{model}} \in [0, 1]$ quantifies the extent to which r answers sub-question q_j.

Execution-Completion Gap (ECG). To diagnose the alignment between the agent's planning capability $\mathcal{I}_{\text{plan}}$ and its final response quality $\mathcal{I}_{\text{resp}}$, we define the following disparity gap:

$$\text{ECG} = \underbrace{\mathcal{A}(\text{TCS}, \text{MTC})}_{\mathcal{I}_{\text{plan}}} - \underbrace{\mathcal{A}(\text{AR}, \text{UIC})}_{\mathcal{I}_{\text{resp}}} \quad (8)$$

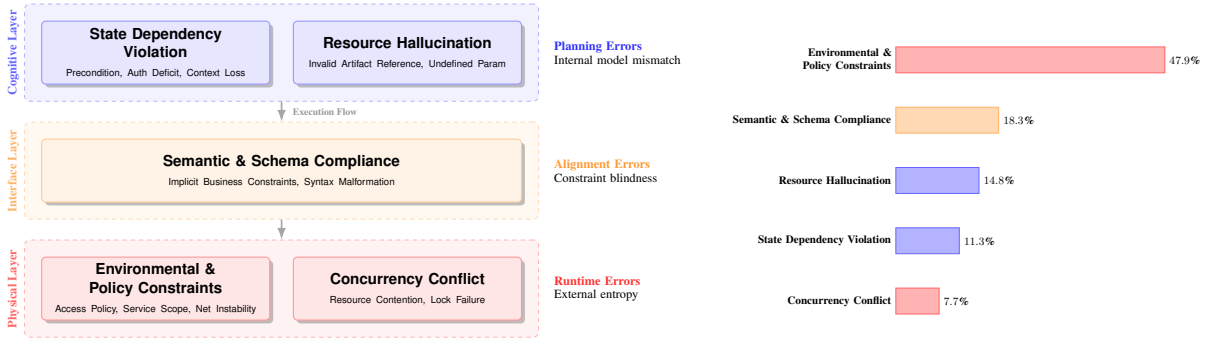
where $\mathcal{A}(\cdot)$ denotes an aggregation operator. A positive divergence ($\text{ECG} \gg 0$) reveals a process-outcome disconnect, implying that the agent correctly invokes tools but fails to effectively synthesize the retrieved information into a valid response.

C Empirical Taxonomy of Tool Execution Failures

To rigorously assess the robustness of MCP Agents in real-world environments, we analyzed execution traces from 576 metamorphic test cases. As shown in Table 6, we classified failure modes into five distinct dimensions. Furthermore, we synthesized these categories into a Layered Failure Model (Figure 7a), which stratifies errors from low-level environmental constraints to high-level cognitive hallucinations. The frequency distribution (Figure 7b) reveals that environmental factors constitute the primary bottleneck, accounting for nearly half of all failures.

Table 6: Taxonomy of Tool Execution Failures in MCP Agents: Categories, Definitions, and Empirical Evidence

Primary Category	Sub-category	Root Cause	Representative Log Trace
Resource Hallucination	*Invalid Artifact Reference*	Reference to a local artifact that has not been materialized by prior execution steps.	`Local image file not found: /tmp/chart.png`
	Undefined Parameter Key	Generation of parameter keys that are absent from the tool's interface schema.	`road traffic search failed: road_name error`
State Dependency Violation	*Precondition Violation*	Invocation of tools without satisfying mandatory initialization states.	`Error: No open pages available.`
	Authentication Deficit	Attempting protected operations without establishing an active user session.	`AuthenticationError: Must be logged in...`
	Context Invalidation	Loss of execution context caused by premature asynchronous navigation logic.	`Execution context was destroyed...`
Environmental & Policy Constraints	*Access Policy Rejection*	Denial of service due to protocol compliance (e.g., robots.txt) or anti-bot verification.	`autonomous fetching is not allowed Sina Visitor System... window.use_fp`
	Service Scope Limitation	Valid requests falling outside the service provider's operational coverage or privileges.	`This area is not supported temporarily INSUFFICIENT_ABROAD_PRIVILEGES`
	Network Instability	Transient failures in the transport layer (DNS resolution, TCP connection).	`getaddrinfo EAI_AGAIN socket hang up`
	Asynchronous Indeterminacy	Operations concluding with timeouts or unknown result states.	`Publish completion timeout`
Concurrency Conflict	*Resource Contention*	Failure to acquire locks on exclusive system resources during parallel execution.	`Browser is already in use... use -isolated`
Semantic & Schema Compliance	*Implicit Business Constraint*	Syntactically valid inputs violating unstated business logic (e.g., length, range limits).	`Title must be 20 characters or less Walking failed: OVER_DIRECTION_RANGE`
	Syntactic Malformation	Structural errors in data formatting preventing successful parsing.	`JSONDecodeError: Invalid control character`



(a) The Layered Failure Model of MCP Agents

(b) Distribution of Failure Modes

Figure 7: Empirical Analysis of Tool Execution Failures. (a) A layered model stratifying errors from physical to cognitive levels. (b) Error frequency distribution, with colors corresponding to their respective layers in (a).

C.1 Physical Layer: Environmental and Policy Friction

The base of the failure stack (Figure 7a, Bottom) represents the collision between the agent's operations and the external system's boundaries. Notably, Environmental & Policy Constraints constitute the largest failure category (47.9%, Figure 7b).

It is imperative to clarify that this high friction rate is not an artifact of an inadequately configured benchmark. As detailed in Appendix D, our environment provides valid authentication tokens for all relevant MCP servers and equips agents with advanced browser automation tools to navigate anti-bot mechanisms. Consequently, rather than a limitation of the benchmark setup, these failures accurately reflect the real-world complexities of the MCP Tool ecosystem, exposing critical deficiencies in agents' environmental awareness and dynamic execution capabilities.

- Protocol-Goal Conflict: Failures such as *Access Policy Rejection* (e.g., `robots.txt`) highlight a fundamental misalignment between agent objectives and external system boundaries. Even when equipped with advanced browser automation tools capable of navigating anti-bot mechanisms, the agent's planner often generates syntactically valid yet contextually unviable execu-

tion plans. When an action is denied—whether enforced by web governance protocols or triggered by server-side anomalies—agents fail to accurately perceive environmental constraints and dynamically replan. Ultimately, this outcome underscores two critical root causes: the agents' inadequate environmental perception capabilities and the inherent operational defects of certain MCP servers.

- Statelessness vs. Defensive Architectures: The prevalence of authentication failures and anti-bot rejections (Table 6) suggests that despite being provided with valid access credentials, MCP Agents operate as **stateless** clients. They lack the persistent session management required to navigate defensive web architectures (e.g., `login sessions`), rendering them ineffective against stateful systems despite correct logic.

C.2 Interface Layer: The Specification Gap

The Interface Layer governs the data exchange protocols between the agent and external tools. Our analysis attributes 18.3% of failures to Semantic & Schema Compliance. A distinct pattern in this category is the violation of *Implicit Business Constraints*, where tool invocations satisfy the explicit interface definition (e.g., JSON Schema) but fail against server-side validation logic. For instance, the error `Title must be 20 characters or less` (Table 6) demonstrates that while the agent correctly generated a string argument as required by the schema, it exceeded the length limit. This finding indicates that standard API specifications often fail to expose critical operational limits, leaving agents prone to generating requests that are syntactically valid but functionally rejected.

C.3 Cognitive Layer: Planning and State Consistency

This layer addresses failures stemming from the agent's internal reasoning and sequential logic. Collectively, *State Dependency Violation* (11.3%) and *Resource Hallucination* (14.8%) account for over a quarter of all errors, reflecting fundamental challenges in synchronizing the generated plan with the dynamic execution environment.

- Violation of Sequential Preconditions: *State Dependency Violations* occur when agents execute operations without satisfying mandatory prerequisites. Our analysis indicates this is frequently driven by *Context Invalidation*, where the agent fails to track dynamic state changes—such as session expiry or asynchronous navigation—across multi-step workflows, leading to out-of-order execution.

- Invalid Resource Referencing: Distinct from semantic inaccuracies in open-ended generation, *Resource Hallucination* in tool use manifests as explicit references to non-existent system artifacts. Errors such as `road_name error` (Table 6) reveal a specific failure mode where the agent attempts to access undefined parameters or files. This points to a synchronization error where the agent's assumption of resource availability diverges from the actual system state.

D Detailed Server Configurations

This appendix provides a comprehensive specification of the 14 MCP servers integrated into our experimental framework. Table 7 presents a structured catalog of these servers, detailing their source repositories, domain categories, and official functional descriptions. To facilitate reproduction and deployment, Figure 8 illustrates the complete, unified JSON configuration used to integrate all servers simultaneously. Further documentation and source code for each implementation are available in their respective repositories (see server names in Table 7: ama; bai; air; hot; xhs; bil; nba; bal; Fin; pla; web; seq; tim; fet).

E Prompts for SGVEF-LOOP

This section details the prompt templates utilized within the SGVEF-LOOP framework. In the interest of transparency and reproducibility, we provide the specific system instructions tailored for each functional module. The prompts are systematically categorized into three domains, corresponding to the framework's core workflow:

- Progressive Exploration & Generation: This category encompasses prompts driving the coverage-guided generation process, extending from initial node-level tool usage (*Initial Problem Generation-Node*) to complex edge-level transitions (*Initial Problem Generation-Edge*). To validate the efficacy of this mechanism, prompts for two ablation variants (*SGVE*, *GVE*) are also included. Furthermore, it covers the generation of fact-augmented metamorphic pairs and the injection of external knowledge via the *Test Case Auditor*.

Table 7: Catalog of MCP Servers Utilized in Experiments.

Domain	Server Name	Functional Description
Travel	AMap Map(ama)	MCP Server for the AMap Map API.
	Baidu Map(bai)	Baidu Map MCP Server is a fully MCP-compliant, open-source Location-Based Service (LBS) solution, providing a comprehensive suite of geospatial APIs and tools for developers and AI agents.
	Airbnb(air)	A comprehensive Desktop Extension for searching Airbnb listings with advanced filtering capabilities and detailed property information retrieval.
Social Media	HotNews(hot)	A Model Context Protocol server that provides real-time hot trending topics from major Chinese social platforms and news sites.
	Xiaohongshu(xhs)	A Model Context Protocol server and CLI tool for Xiaohongshu. Powered by Puppeteer, it supports automation capabilities such as login, publishing, searching, and browsing recommendations.
	Bilibili(bil)	This is a Bilibili video search server based on the Model Context Protocol. The server provides a simple API interface that allows users to search for video content on Bilibili. It includes LangChain usage examples and test scripts.
Sports	NBA Stats(nba)	Access comprehensive NBA statistics via Model Context Protocol.
	Balldontlie(bal)	An MCP Server implementation that integrates the Balldontlie API, to provide information about players, teams and games for the NBA, NFL and MLB.
Finance	Yahoo Finance(Fin)	A simple MCP server for Yahoo Finance using yfinance. This server provides a set of tools to fetch stock data, news, and other financial information.
Browser	Playwright(pla)	A Model Context Protocol (MCP) server that provides browser automation capabilities using Playwright. This server enables LLMs to interact with web pages through structured accessibility snapshots, bypassing the need for screenshots or visually-tuned models.
	Web Search(web)	A Model Context Protocol server that enables free web searching using Google search results, with no API keys required.
Utilities	Sequential Thinking(seq)	A MCP server implementation that provides a tool for dynamic and reflective problem-solving through a structured thinking process.
	Time(tim)	A Model Context Protocol server that provides time and timezone conversion capabilities. This server enables LLMs to get current time information and perform timezone conversions using IANA timezone names, with automatic system timezone detection.
	Fetch(fet)	A Model Context Protocol server that provides web content fetching capabilities. This server enables LLMs to retrieve and process content from web pages, converting HTML to markdown for easier consumption.

- Verification: This category comprises prompts designed to ensure the structural and semantic validity of test cases. Specifically, it involves the *Test Case Refiner*, which addresses semantic consistency and parameter solvability.
- Evaluation: This category contains prompts that direct the evaluator agents to assess tool chain consistency and response quality, specifically targeting *Answer Relevance* and *Unified Intent Coverage (UIC)*.

The full content of these prompts is presented below.

Figure 8: Unified JSON configuration for integrating all 14 MCP servers.

```
{
  "mcpServers": {
    "amap-maps": {
      "command": "npx",
      "args": [
          "-y",
          "@amap/amap-maps-mcp-server"
      ],
      "env": {
          "AMAP_MAPS_API_KEY": ""
      }
    },
    "baidu-maps": {
      "command": "python",
      "args": ["-m", "mcp_server_baidu_maps"],
      "env": {
        "BAIDU_MAPS_API_KEY": "<YOUR_API_KEY>"
      }
    },
    "airbnb": {
      "command": "npx",
      "args": [
        "-y",
        "@openbnb/mcp-server-airbnb"
      ]
    },
    "mcp-server-hotnews": {
      "command": "npx",
      "args": [
        "-y",
        "@wopal/mcp-server-hotnews"
      ]
    },
    "xhs-mcp": {
      "command": "npx",
      "args": ["xhs-mcp", "mcp"],
      "env": { "XHS_ENABLE_LOGGING": "true" }
    },
    "bilibili-search": {
    "command": "npx",
    "args": ["bilibili-mcp-js"],
    "description": "Bilibili Video Search MCP service, enabling AI applications to search
        Bilibili video content."
    },
    "nba-stats": {
      "command": "uvx",
      "args": ["nba-stats-mcp"]
    },
    "balldontlie": {
      "command": "npx",
      "args": [
        "-y",
        "balldontlie-mcp"
      ],
      "env": {
        "BALLDONTLIE_API_KEY": "YOUR API KEY HERE"
      }
    },
    "yfmcp": {
      "command": "uvx",
      "args": ["yfmcp@latest"]
    },
    "playwright": {
      "command": "npx",
      "args": [
        "@playwright/mcp@latest"
      ]
    },
    "web-search": {
      "command": "node",
      "args": ["/path/to/web-search/build/index.js"]
    },
    "sequential-thinking": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-sequential-thinking"
      ]
    },
    "time": {
      "command": "uvx",
      "args": ["mcp-server-time"]
    },
    "fetch": {
      "command": "uvx",
      "args": ["mcp-server-fetch"]
    }
  }
}
```

Initial Problem Generation (Node)

You are an expert in advanced question design, logic analysis, and fact-checking. Your task is to generate a user's initial query that meets strict standards based on a randomly selected set of tools, and to design a fact-checking/disambiguation query suitable for Wikipedia based on that problem. This serves to facilitate subsequent fact-finding on Wikipedia to optimize the initial query.

Analysis Goals:

1. Logical Chaining (Data Flow): Carefully check if the output of one tool among these can naturally serve as the input for another (based on description and parameters).
2. Reasonable Combination (Context): The functions of these tools must be capable of being called consecutively or simultaneously within the same real-world scenario.
3. Redundancy & Conflict: If tool functions are completely repetitive or mutually contradictory, they are considered unable to form a valid problem.

Success Criteria: Success is achieved only when these tools can jointly solve a complex, non-single-step real-world user request through data flow chaining or contextual logical combination.

Provided Tools (JSON, including name, description, parameters): {tools_json_str}

Instructions:

1. Primarily determine if this set of tools meets the success criteria.
2. If judged as successful, generate a complex natural language request that a real user would ask, which must meet the following:
   - A single, coherent real-world scenario.
   - Must require at least 2 tools working together to answer completely.
   - Specific past dates are not allowed (unless discussing famous historical events). Time expressions should be vague, such as "today," "recently," "this weekend," or "in the near future."
   - If the question involves expressions like "my home," it must be replaced with a specific real location like "Zhongguancun, Haidian District, Beijing."
   - The question cannot involve private information, future predictions, model inferences, etc.
   - The request must be complex, containing at least multi-step reasoning or operations.
3. Generate a fact-checking query for this request that is suitable for a Wikipedia search and helps disambiguate or optimize the initial_problem. For example:
   - Background/usage of core entities.
   - Definitions or distinctions of key concepts.
   - Encyclopedic information on real locations, organizations, people, or events.
   - (Do not generate content unavailable on Wikipedia, such as private information, future predictions, model inferences, etc.)
4. Output strict JSON, without adding any explanation or extra text:

   ```
   {
     "initial_problem": "The complex user query text you generated",
     "fact_seek_query": "The fact/disambiguation query text for the core entity or concept"
   }
   ```

5. If judged as a failure (tools cannot be reasonably combined), return only a single word: null

Initial Problem Generation (Edge)

You are an expert in advanced question design, logic analysis, and fact-checking. Your task is to generate an initial user query that meets strict standards based on a randomly drawn set of tools, and to design a fact-checking/disambiguation query suitable for Wikipedia based on that problem. This serves to facilitate subsequent retrieval of relevant factual information on Wikipedia to optimize the initial query.

Analysis Goals:

1. Logical Chaining (Data Flow): Carefully check if the output of one tool among these can naturally serve as the input for another (judging by 'description' and 'parameters').
2. Reasonable Combination (Context): The functions of these tools must be capable of being called consecutively or simultaneously within the same real-world scenario.
3. Redundancy & Conflict: If tool functions are completely repetitive or mutually contradictory, they are considered unable to form a valid problem.

Success Criteria:
Success is achieved only when these tools can jointly solve a complex, non-single-step real-world user request through "data flow chaining" or "contextual logical combination."

Provided Tools (JSON, including name, description, parameters): {tools_json_str}
Feature categories and corresponding tool lists: {feature_buckets}
Existing feature paths: {feature_path_set}.
Please prioritize generating feature paths that have not appeared yet to improve tool transition coverage.

Instructions:

1. First, determine if this set of tools meets the success criteria.
2. If judged as successful, generate a complex natural language request that a real user would ask, which must meet the following:
   - A single, coherent real-world scenario.
   - Specific past dates are not allowed (unless discussing famous historical events). Time expressions should be vague, such as "today," "recently," "this weekend," or "in the near future."
   - If the question involves expressions like "my home," it must be replaced with a specific real location.
   - The question cannot involve private information, future predictions, model inferences, etc.
   - The request must be complex, containing at least multi-step reasoning or operations.
   - Prioritize generating feature paths that have not appeared yet to improve feature path coverage.
3. Generate a fact-checking query for this request that is suitable for a Wikipedia search and helps disambiguate or optimize the 'initial_problem'. For example:
   - Background/usage of core entities.
   - Definitions or distinctions of key concepts.
   - Encyclopedic information on real locations, organizations, people, or events.
   - (Do not generate content unavailable on Wikipedia, such as private information, future predictions, model inferences, etc.)
4. Output strict JSON, without adding any explanation or extra text:

```
{
    "initial_problem": "The complex user query text you generated",
    "fact_seek_query": "The fact/disambiguation query text for the core entity or concept"
}
```

5. If judged as a failure (tools cannot be reasonably combined), return only a single word: null

Initial Problem Generation (SGVE)

You are a senior expert in question design, logic analysis, and fact-checking. Your task is to generate an initial user query meeting strict standards based on a randomly selected set of tools, and to design a fact-checking/disambiguation query suitable for Wikipedia based on that problem. This serves to facilitate subsequent fact-finding on Wikipedia to optimize the initial query.

Analysis Goals:

1. Logical Chaining (Data Flow): Carefully check if the output of one tool can naturally serve as the input for another (judging by description and parameters).
2. Reasonable Combination (Context): The functions of these tools must be capable of being called consecutively or simultaneously within the same real-world scenario.
3. Redundancy & Conflict: If tool functions are completely repetitive or mutually contradictory, they are considered unable to form a valid problem.

Success Criteria:
Success is achieved only when these tools can jointly solve a complex, non-single-step real-world user request through "data flow chaining" or "contextual logical combination."

Provided Tools (JSON, including name, description, parameters): {tools_json_str}
Feature categories and corresponding tool lists: {feature_buckets}

Instructions:

1. First, determine if this set of tools meets the success criteria.
2. If judged as successful, generate a complex natural language request that a real user would ask, which must meet the following:
   - A single, coherent real-world scenario.
   - Specific past dates are not allowed (unless discussing famous historical events). Time expressions should be vague, such as "today," "recently," "this weekend," or "in the near future."
   - If the question involves expressions like "my home," it must be replaced with a specific real location.
   - The question cannot involve private information, future predictions, model inferences, etc.
   - The request must be complex, containing at least multi-step reasoning or operations.
   - Prioritize generating feature paths that have not appeared yet to improve feature path coverage.
3. Generate a fact-checking query for this request that is suitable for a Wikipedia search and helps disambiguate or optimize the initial_problem. For example:
   - Background/usage of core entities.
   - Definitions or distinctions of key concepts.
   - Encyclopedic information on real locations, organizations, people, or events.
   - (Do not generate content unavailable on Wikipedia, such as private information, future predictions, model inferences, etc.)
4. Output strict JSON, without adding any explanation or extra text:

   ```
   {
     "initial_problem": "The complex user query text you generated",
     "fact_seek_query": "The fact/disambiguation query text for the core entity or concept"
   }
   ```

5. If judged as a failure (tools cannot be reasonably combined), return only a single word: null

Initial Problem Generation (GVE)

You are an Agent test case generator. For Agents under the MCP paradigm, generate test cases that satisfy Real-world Executability. Each test case must be a complex real-world task capable of invoking 3 to 6 tools.

Currently, the intelligent Agent supports {FEATURE_COUNT} functional tools.
Detailed descriptions and required parameters for these tools: {FEATURE_BUCKETS_ENRICHED}
Output Format: A JSON array.

[Generation Requirements]

- Please generate 5 sets of metamorphic pairs at once. Each set contains 2 test cases, totaling 10 complex test cases.
- The user_request of the 2 test cases in the same set must be strictly semantically consistent, but the wording/phrasing/order/details should differ.
- Test cases should be complex real-world tasks capable of calling 3 to 6 tools.
- Fields for each test case:
  - id: 1, 2
  - user_request: Natural language task(complex task fitting real-life scenarios, must not contain tool names, involving 3-6 tool calls).
  - expected_tool_chain: [tool1, tool2, tool3,...], array of tool names, strictly in calling order.
  - Test cases require diversity (different tools, different scenarios, different details, different phrasing styles, different parameters), but must be realistically executable.
  - Generated test cases cannot repeatedly call the same tools; diversity is required.
  - Output ONLY a JSON array, do not use Markdown code fences or extra comments.
  - Tool names must strictly come from the {TOOL_LIST} list.

[Semantic Equivalence Hint (Metamorphic Pairs)]
Allow substitution of synonymous expressions, syntactic adjustments, and addition of context that does not affect intent, but intent and constraints must remain consistent (critical information like time/location/budget/travel mode, etc., must be consistent).

Generate Fact Augmented Testcase

You are an AI Agent tool chain planning and fact-augmented test case generation expert.Now you are given three inputs:

Provided Inputs:

1. Selected tools (JSON): {tools_json_str}
2. Original Initial Problem: {initial_problem}
3. Relevant facts retrieved from vector database (Facts): {facts_text}

Your task is to perform fact augmentation on the {initial_problem} based on the above facts (if the relevance of the facts is insufficient, i.e., less than 0.9, do not perform fact augmentation), and generate a pair of semantically consistent test cases for subsequent metamorphic testing.
The two cases must be completely equivalent in semantics/intent (i.e., user intent must remain consistent), but must differ in phrasing (e.g., rewriting, synonym substitution, changing parameter order, unit conversion, detail expansion/merging, etc.) to verify the system's robustness to equivalent inputs.

- Each case must be clearer and unambiguous;
- Each case must contain no factual errors;
- Each case must enhance the entities, background, and context of the request;
- Each case must stably trigger at least 3 tools;
- All descriptions in each case must conform to known real-world facts.

Finally, output only a "strict JSON array" containing exactly two test cases, formatted as follows (do not add any extra explanation, do not include superfluous text, do not use Markdown):

```
[
  {
    "id": 1,
    "user_request": "Final user request rewritten based on facts",
    "expected_tool_chain": ["tool1", "tool2", ...]
  },
  {
    "id": 2,
    "user_request": "Another equivalent request with different phrasing",
    "expected_tool_chain": ["tool1", "tool2", ...]
  }
]
```

If valid test cases cannot be generated, return "null".

Test Case Auditor

You are an AI Agent Test Case Strict Audit Expert driven by a large language model. Your task is to audit a pair of "Metamorphic Test Pairs", which includes two natural language requests (Request A and Request B) that must point to the same Expected Tool Chain (Expected Tool Chain).
Please perform logical verification on the input based on the provided [Tool Definitions (Tool Definitions)] and return the result in strict JSON format.

Scoring Rubric - Max Score 100

1. Semantic Equivalence:
   - Request 1 and Request 2 must express exactly the same intent and constraints.
   - If either party lacks key information resulting in the inability to trigger the same tool chain, it is considered inconsistent.
2. Parameter Completeness & Data Flow:
   - Iterate through each tool in the tool chain to check the source of its [Required Parameters].
   - The source must be one of the following two: (a) Explicitly provided by the user: Directly included in the Request text. (b) Implicitly derived from context: Can be logically derived from the output fields of Predecessor Tools (e.g., a predecessor tool outputs "latitude and longitude", and the current tool requires "coordinates").
   - Error Determination: If a required parameter is not mentioned in the Request and cannot be obtained from the output of predecessor tools, mark it as missing_params.

Output Format (Strict JSON).

```
{
    "score": 0-100, // Integer score. >=80 is considered a Pass, <80 is considered a Fail
    "issues": [
        {
            "case": 1 | 2,
            "tool": "string",
            "error_type": "Missing Parameter" | "Semantic Mismatch" | "Hallucination",
            "details": "Detailed explanation of the reason for deduction",
        }
    ],
    "suggestions": ["Specific, actionable modification suggestions guiding how to fix the above issues by modifying the natural language request. If score >= 80, this array can be left empty."]
}
```

Test Case Refiner

You are a professional Test Corpus Repair & Polishing Expert. You will receive a pair of defective test cases (Request 1 and Request 2) and the [Modification Suggestions] provided by the auditor.Your task is to rewrite these two requests so that they meet logical requirements while remaining natural and fluent.

Core Tasks

1. Fix Defects: Based on the modification suggestions, naturally integrate missing parameter information (such as time, location, quantity) into the sentences.
2. Maintain Metamorphic Relationship:
   - The core semantics of Request 1 and Request 2 must be exactly the same (ensuring the same tool chain can be triggered).
   - The phrasing styles of Request 1 and Request 2 must remain different (e.g., A uses formal declarative sentences, B uses colloquial questions; or A states the background before the request, B gets straight to the point).

Strict Constraints (Strict Constraints) - Violation will render the test invalid!

1. No Leaking Tool Names: Absolutely do not mention specific tool names, API names, or function names in the natural language request (e.g., Forbidden to say "Please call ToolWeatherAPI to query...", Must say "Check the weather for me...").
2. No Modifying Tool Chain: Your rewrite cannot change the originally expected tool invocation logic.
3. No Mechanical Stacking: Do not rigidly insert parameters; speak like a real user.

Output Format (Strict JSON)

```
{
    "rewritten": [
        "Rewritten and polished Request 1 (Natural Language)",
        "Rewritten and polished Request 2 (Natural Language, phrased differently)"
    ],
    "reasoning": "Briefly explain how you integrated missing information into natural language and maintained the difference between the two"
}
```

Evaluator (Answer Relevance-Forward Judge)

Please evaluate the "Faithfulness" and "Usefulness" of the Agent's response.

- [User Request]: {request}
- [Tool Execution Context] (The Agent executed the following tools and obtained these results): {tool_outputs}
- [Agent Response]: {response}

Scoring Rubric (Please provide a precise score between 0.0 and 1.0):

1. [0.0 - 0.2] Complete Failure:
   - The response is completely irrelevant to the question.
2. [0.3 - 0.4] Severe Defects:
   - Attempts to answer but severely misinterprets the data output by the tools (e.g., numerical errors, reversed objects);
   - Contains critical factual Hallucinations.
3. [0.5 - 0.6] Partially Valid (Passing Line):
   - Answers the core conclusion of the question but misses important constraints;
   - Contains a small amount of unverified redundant information, but it does not affect the core judgment.
4. [0.7 - 0.8] Good:
   - The core conclusion is correct and faithful to the tool output;
   - Lacks slightly only in completeness or precision (e.g., numerical values not keeping decimal places, or missed secondary attribute information).
5. [0.9 - 1.0] Perfect:
   - Based on the tool execution results, answers all user demands precisely, completely, and logically;
   - No hallucinations, no omissions.

Please return strict JSON format:

{ "score": <float>, "reason": "<short reason for scoring>" }

Evaluator (UIC-Intent Planner)

Part 1: UIC_PLAN
Please analyze the following two [Semantically Equivalent] user requests (Request A and Request B).
Your task is:

1. Extract their common Core Intent.
2. Decompose the core intent into several independently verifiable Atomic Sub-intents to serve as criteria for subsequent evaluation.

- [Request A] {a}
- [Request B] {b}

Please return strictly in JSON format:

```
{
    "unified_intent": "One sentence summarizing the common goal",
    "intents": [
        { "id": "1", "desc": "Description of sub-intent 1 (e.g., Query the specific real-time temperature in Beijing)" },
        { "id": "2", "desc": "Description of sub-intent 2 (e.g., Confirm wind scale)" }
    ]
}
```

Part 2: UIC_JUDGE
Please evaluate the coverage of the [Agent Response] item by item based on the following [Sub-intent List].

- [Sub-intent List] {intents}
- [Agent Response] {resp}

Scoring Criteria:

- 1.0∼0.7 (Hit): Completely and clearly answered the question of the sub-intent, or provided a reasonable explanation based on tool execution results (including reasonable refusal due to "no data found").
- 0.7∼0.3 (Partial): Only vaguely mentioned, or answered the question but missed key details (e.g., missing units, insufficient precision).
- 0.3∼0.0 (Miss): Completely unmentioned, or the answer is completely irrelevant/wrong regarding the sub-intent.

Please return strictly in JSON format:

```
{
    "scores": {
        "1": 1.0,
        "2": 0.5
    }
}
```