From Tool Selection to Autonomous Execution: Process Templates, Sandboxed Code, and Modular Orchestration for Reliable LLM Agents at Scale

Anonymous authors
Paper under double-blind review

Abstract

Large language model (LLM) agents must route and execute tools from large, overlapping libraries. Prior work reported routing degradation under flat catalogs; we study both routing and execution under scale, noise, and multi-step control. Across synthetic catalogs ($N = 100 \ldots 1000$) and a 59-skill real-world library, expert disambiguating descriptions improve routing accuracy for some models (e.g., Gemini 3.1 Flash Lite +4.6% to +6.3% at $N = 1000$) while hurting others (e.g., Claude 4.5 Haiku −42.5% at $N = 1000$), demonstrating that description benefits are model-dependent rather than universal. For execution, atomic benchmarks reveal a two-stage failure pattern: bare prompts fail primarily due to JSON schema collapse, while standard operating procedure (SOP) text alone suffers arithmetic failures (0.0% end-to-end success on complex math). Coupling expert SOPs with Python sandbox execution resolves schema errors and restores calculation fidelity to 61.1%–100.0% across four core models on synthetic atomic tasks; however, production file-based tasks remain challenging (11.1%–33.3% E2E success). Robustness stress tests show that mild noise (typos, fluff) is well-tolerated, but explicit adversarial instruction injection collapses direct execution (0.0% success), which an AST-based script-integrity guardrail partially recovers (16.7%–46.7%). In a 30-task live multi-step Data Scientist stress test, partitioned Expert stages ($N \leq 15$ tools per stage) achieve 93.3%–100.0% E2E completion versus 46.7%–76.7% for flat Bare catalogs, with significantly higher retry loopbacks under Bare exposure. Together, these results support a practical agent design recipe—expert templates, sandbox execution, stage-wise tool partitions, and light guardrails—without claiming a single universal cosine phase boundary or infinite linear scalability.

1 Introduction

Autonomous software agents powered by large language models (LLMs) are increasingly deployed to execute multi-step workflows across complex domain toolkits (Schick et al., 2023; Qin et al., 2023; Li et al., 2023). As tool libraries expand to hundreds or thousands of functions, agents face two primary failure modes: routing collapse, where semantic overlap among tool descriptions confuses tool selection, and execution breakdown, where valid tool calls fail due to schema non-compliance, parameter hallucinations, or invalid code outputs (Anonymous, 2026).

Prior literature has emphasized tool-selection collapse in flat catalog representations (Anonymous, 2026). However, evaluating routing in isolation overlooks how tool selection interacts with downstream execution and multi-step orchestration. In real-world enterprise environments, an agent must not only choose the correct tool from a dense catalog but also synthesize valid parameters, handle file-based data streams, execute multi-step logic without compounding errors, and withstand noisy or adversarial inputs.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

In this work, we present a systematic empirical investigation bridging tool routing, atomic execution, guardrailed execution, and multi-step orchestration across synthetic and real-world skill libraries. Our study spans model families including DeepSeek (V4 Flash, V4 Pro), OpenAI (GPT-4o-mini, GPT-OSS 20B), Google (Gemini 3.1 Flash Lite, Gemma-4 26B), Anthropic (Claude 4.5 Haiku), and NVIDIA (Nemotron-3-Ultra 550B).

Core Contributions:

1. Routing vs. Description Quality: We evaluate routing across synthetic ($N = 100 \dots 1000$) and real-world 59-skill catalogs, showing that expert disambiguating descriptions provide model-dependent gains rather than universal fixes.

2. Atomic Execution & Code Sandboxing: We isolate a two-stage execution failure taxonomy—schema collapse vs. arithmetic failure—and demonstrate that pairing Standard Operating Procedures (SOPs) with sandboxed Python code execution restores calculation fidelity from 0.0% to 100.0% on clean atomic queries.

3. Guardrails & Noise Robustness: We evaluate an L0–L5 noise gradient (typos, fluff, contradictions, adversarial prompt injections). While direct execution collapses to 0.0% under L5 injection, an Abstract Syntax Tree (AST) script-integrity guardrail recovers up to 46.7% of failed runs.

4. Live Multi-Step Orchestration: On a live 30-task Data Scientist benchmark, stage-partitioned tool exposure ($N \leq 15$ tools per stage) achieves 93.3%–100.0% end-to-end completion compared to 46.7%–76.7% under flat unpartitioned catalogs.

2 Related Work

Tool Selection & Phase Transitions. The scaling of tool availability in LLM agents has been investigated in benchmarks such as ToolBench (Qin et al., 2023) and API-Bank (Li et al., 2023). Anonymous (2026) identified sharp accuracy drops beyond ~ 60 tools in flat catalogs due to semantic overlap. We extend this work by decoupling description quality from raw catalog size N and analyzing the underlying vector-space crowding.

Code Execution vs. JSON Function Calling. Standard function-calling paradigms rely on JSON schema generation. However, complex mathematical and data-processing tasks suffer from token generation limits and numeric hallucination. Executing code in sandboxed environments (Schick et al., 2023) offloads computation from language generation to interpreted runtimes.

Guardrails & Control-Flow Systems. Production agent systems deploy static analysis and runtime validation to mitigate prompt injection and malformed outputs (Center for AI Safety & Scale AI, 2025). Our study evaluates how lightweight AST checks and dynamic multiple-testing penalties preserve state integrity in iterative workflows.

3 Methodology and System Architecture

3.1 Process Templates and Disambiguating Descriptions

We formalize tool representation into two distinct exposure contracts:

- Bare Descriptions: Concise, single-sentence summaries of tool functionality (e.g., standard API docstrings).

- Expert Process Templates (SOPs): Structured prompt specifications including explicit input/output schemas, domain preconditions, parameter range constraints, and disambiguation notes distinguishing the tool from semantically adjacent distractors.

3.2 Sandboxed Code Execution Engine

To prevent arithmetic hallucinations and JSON format errors, we deploy a closed-loop Python execution sandbox (Figure 1). When an agent invokes a tool under the Expert Sandbox condition, it emits an executable Python script loading task data into isolated subprocess environments. Outputs are passed through structured stdout parsers and validated against task contracts.

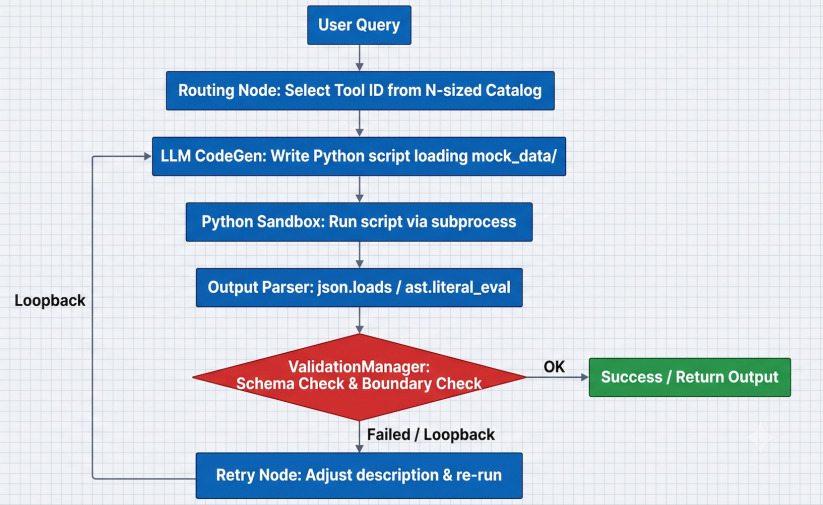

Figure 1: Closed-loop single-skill workflow engine architecture: tool selection, Python script synthesis, sandboxed subprocess execution, and ValidationManager guardrails.

3.3 Guardrails and Production Validation Infrastructure

We implement three validation guardrails:

1. Script-Integrity Guardrail: AST inspection checks for mandatory library imports, prohibited system calls, and required output formatting. Malformed scripts trigger bounded re-prompting with target error feedback.
2. Multiple-Testing Correction Guardrail: To prevent overfitting in iterative backtesting workflows, validation contracts scale up performance thresholds logarithmically:

$$T_{\text{adjusted}} = \min\left(0.95, T_{\text{base}} + 0.04 \times \ln(K)\right) \tag{1}$$

   where K is the trial count and T_{base} is the baseline threshold (0.60–0.70).
3. Dynamic Validation Manager: Adjusts validation weights W_{metric} and drift penalties W_{drift} based on sample size ($N < 100$), class imbalance ($< 30\%$), and data missingness ($> 10\%$).

3.4 Modular Orchestration & Tool Partitioning

For multi-step workflows, flat catalog exposure causes severe error propagation across sequential decision nodes. We implement stage-wise tool partitioning, restricting the local tool view at node i to a small subset $N_i \leq 15$. The theoretical cumulative graph success probability is bounded by:

$$P_{\text{success, graph}} = \prod_{i=1}^{d} (1 - \epsilon_i) \tag{2}$$

where d is graph depth and ϵ_i is the local stage failure rate. Keeping $\epsilon_i \approx 0$ via small stage-local tool sets prevents exponential degradation of end-to-end task completion.

4 Experimental Evaluation and Results

4.1 Phase 1: Routing Accuracy & Description Engineering

We evaluate tool routing across synthetic catalogs ($N \in \{100, 200, 500, 1000\}$) built from 20 target skills surrounded by distractors, as well as a 59-skill real-world dataset.

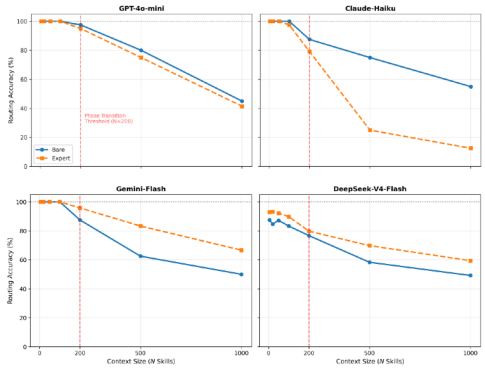

Figure 2: Synthetic routing accuracy vs. library size N.

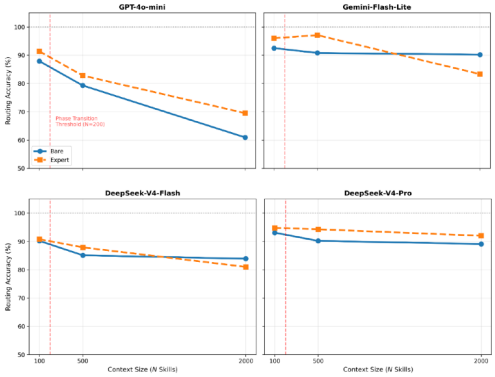

Figure 3: Real-world 59-skill routing accuracy vs. N.

Table 1: Phase 1 Routing Accuracy (%) on Synthetic 20-Skill Library (Bare / Expert).

Model	**N = 100**	**N = 200**	**N = 500**	**N = 1000**
GPT-4o-mini	100.0 / 100.0	95.0 / 90.0	55.0 / 50.0	45.0 / 41.5
Claude 4.5 Haiku	100.0 / 100.0	90.0 / 95.0	60.0 / 30.0	55.0 / 12.5
Gemini 3.1 Flash Lite	100.0 / 100.0	95.0 / 100.0	60.0 / 65.0	50.0 / 66.7
DeepSeek V4 Flash	100.0 / 100.0	90.0 / 95.0	55.0 / 60.0	49.2 / 59.4

As shown in Table 1 and Figure 2, routing performance degrades sharply beyond $N = 200$. However, Expert descriptions yield divergent results across architectures: Gemini 3.1 Flash Lite and DeepSeek V4 Flash gain +16.7% and +10.2% at $N = 1000$, whereas Claude 4.5 Haiku drops by −42.5% due to long-context attention distraction.

Table 2: Phase 1 Routing Accuracy (%) on 59 Real-World Target Skills (Bare / Expert).

Model	**N = 100**	**N = 500**	**N = 1000**
GPT-4o-mini	87.9 / 91.4	79.3 / 82.8	65.5 / 70.7
Gemini 3.1 Flash Lite	92.5 / 96.0	90.8 / 97.1	84.5 / 89.1
DeepSeek V4 Flash	90.2 / 90.8	85.1 / 87.9	78.2 / 81.6
DeepSeek V4 Pro	93.1 / 94.8	90.2 / 94.3	86.2 / 90.8

On real-world skills (Table 2 and Figure 3), Expert descriptions consistently improve selection accuracy by +3.4% to +6.3% across all models, confirming that domain-specific disambiguation helps when prompt size remains within optimal context windows.

4.2 Phase 2: Atomic Execution & Code Sandboxing

We test execution fidelity across clean queries (12 tasks), noisy queries (18 tasks), and production file-based tasks (CSV/JSON manipulation).

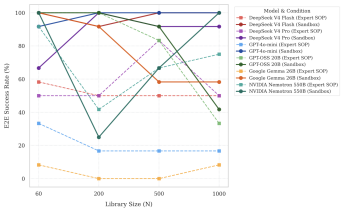

Figure 4: Clean atomic execution vs. N.

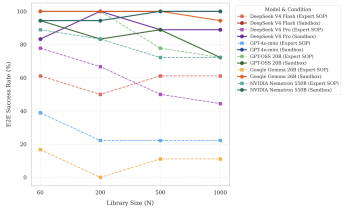

Figure 5: Noisy atomic execution vs. N.

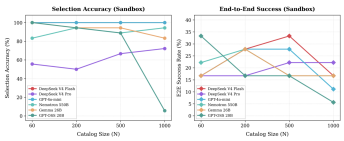

Figure 6: Production file execution vs. N.

Table 3: Atomic Primitives Benchmark on 12 Clean Queries (% Accuracy: Expert / Sandbox).

Model	**N = 60**	**N = 200**	**N = 500**	**N = 1000**
DeepSeek V4 Flash	58.3 / 100.0	50.0 / 91.7	50.0 / 100.0	50.0 / 100.0
DeepSeek V4 Pro	50.0 / 66.7	50.0 / 100.0	83.3 / 91.7	50.0 / 91.7
GPT-4o-mini	33.3 / 91.7	16.7 / 100.0	16.7 / 100.0	16.7 / 100.0
GPT-OSS 20B	100.0 / 91.7	100.0 / 83.3	75.0 / 83.3	66.7 / 66.7
Gemma-4 26B	50.0 / 100.0	33.3 / 91.7	33.3 / 91.7	33.3 / 91.7

Table 3 demonstrates that text-based SOPs without sandbox execution suffer severe calculation drops (GPT-4o-mini falls to 16.7%). Sandboxed Python execution restores accuracy to 91.7%–100.0% across models by offloading arithmetic computation to code.

Table 4: Atomic Primitives Benchmark on 18 Noisy Queries (% Accuracy: Expert / Sandbox).

Model	**N = 60**	**N = 200**	**N = 500**	**N = 1000**
DeepSeek V4 Flash	61.1 / 100.0	50.0 / 100.0	61.1 / 100.0	61.1 / 100.0
DeepSeek V4 Pro	77.8 / 83.3	66.7 / 100.0	50.0 / 88.9	44.4 / 88.9
GPT-4o-mini	38.9 / 94.4	22.4 / 94.4	22.4 / 100.0	22.4 / 100.0
GPT-OSS 20B	100.0 / 94.4	100.0 / 83.3	77.8 / 88.9	72.2 / 72.2
Gemma-4 26B	55.6 / 94.4	38.9 / 88.9	38.9 / 88.9	38.9 / 88.9

Under query noise (Table 4), the sandbox baseline maintains near-perfect performance (88.9%–100.0%).

However, production file tasks (Table 5) reveal that pure text prompts achieve near 0.0% success, and even sandboxed execution reaches only 11.1%–33.3%, illustrating that file I/O and complex data parsing require dynamic feedback loops.

4.3 Phase 3: Robustness & Fracture-Point Stress Tests

We test an adversarial noise gradient L0 (clean) to L5 (prompt injection) at $N = 500$.

Table 6 and Figure 7 highlight that while L5 prompt injections reduce direct execution to 0.0% across most models, the AST script-integrity guardrail recovers 46.7% of executions for DeepSeek V4 Flash, GPT-4o-mini, and Nemotron-3-Ultra.

Table 7 confirms high robustness across novelty and ambiguity variations when tool descriptions are clearly structured.

Table 5: Production-Grade File-Based Benchmark (% Accuracy: Expert / Sandbox).

Model	**N = 60**	**N = 200**	**N = 500**	**N = 1000**
DeepSeek V4 Flash	0.0 / 16.7	0.0 / 27.8	0.0 / 33.3	0.0 / 16.7
DeepSeek V4 Pro	0.0 / 16.7	0.0 / 16.7	0.0 / 22.2	0.0 / 22.2
GPT-4o-mini	5.6 / 16.7	11.1 / 27.8	22.2 / 27.8	5.6 / 11.1
GPT-OSS 20B	11.1 / 33.3	11.1 / 16.7	11.1 / 16.7	11.1 / 5.6
Gemma-4 26B	0.0 / 11.1	0.0 / 16.7	0.0 / 16.7	0.0 / 11.1

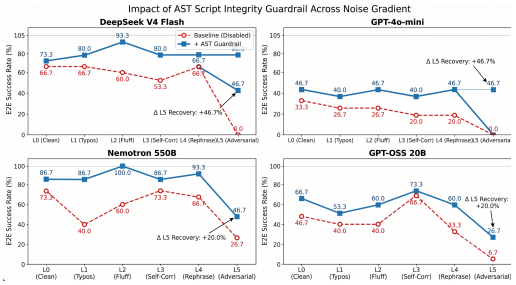

Figure 7: Noise gradient L0–L5 robustness.

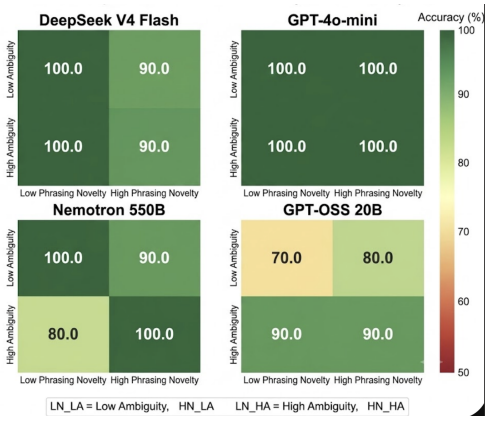

Figure 8: Multi-role volume stress simulation.

Table 6: Robustness Under Noise Gradient (L0–L5) & AST Guardrail Recovery (%).

Model	L0 (Clean)	L1 (Typo)	L2 (Fluff)	L3 (Contra)	L4 (Implicit)	L5 (Injection: Direct/AST)
DeepSeek V4 Flash	66.7 / 73.3	66.7 / 80.0	60.0 / 93.3	53.3 / 80.0	66.7 / 80.0	0.0 / 46.7
GPT-4o-mini	33.3 / 46.7	26.7 / 40.0	26.7 / 46.7	20.0 / 40.0	20.0 / 46.7	0.0 / 46.7
Nemotron-3-Ultra 550B	73.3 / 86.7	40.0 / 86.7	60.0 / 100.0	73.3 / 86.7	66.7 / 93.3	26.7 / 46.7
GPT-OSS 20B	66.7 / 80.0	60.0 / 73.3	53.3 / 80.0	46.7 / 73.3	46.7 / 73.3	0.0 / 16.7

Table 7: Query Variation Matrix — Routing Accuracy (%) across Quadrants.

Model	Low Amb / Low Nov	Low Amb / High Nov	High Amb / Low Nov	High Amb / High Nov
GPT-4o-mini	100.0%	100.0%	100.0%	100.0%
DeepSeek V4 Flash	100.0%	90.0%	100.0%	90.0%
Nemotron-3-Ultra 550B	100.0%	90.0%	80.0%	100.0%
GPT-OSS 20B	70.0%	80.0%	90.0%	90.0%

4.4 Phase 4: Live Multi-Step Data Scientist Orchestration

We execute a 30-task live multi-step benchmark comparing stage-partitioned Expert exposure ($N \leq 15$) against unpartitioned flat Bare catalogs.

Table 8: Empirical Data Scientist Orchestrator Results (30 Live API Tasks).

Model & Exposure	E2E Success	Direct Success	Loopback Recovery	Avg Loopbacks
DeepSeek V4 Flash (Expert)	100.0%	100.0%	N/A	0.00
DeepSeek V4 Flash (Bare Flat)	63.3%	53.3%	21.4%	0.87
GPT-4o-mini (Expert)	100.0%	100.0%	N/A	0.00
GPT-4o-mini (Bare Flat)	46.7%	46.7%	0.0%	1.07
Nemotron-3-Ultra 550B (Expert)	93.3%	60.0%	83.3%	0.70
Nemotron-3-Ultra 550B (Bare Flat)	46.7%	16.7%	36.0%	1.57
GPT-OSS 20B (Expert)	93.3%	86.7%	50.0%	0.23
GPT-OSS 20B (Bare Flat)	76.7%	66.7%	30.0%	0.57

Table 8 provides decisive evidence: stage-wise tool partitioning achieves 93.3%–100.0% E2E completion, while flat catalogs collapse to 46.7%–76.7% with high loopback retry costs. Notably, GPT-4o-mini under flat exposure fails to recover from a single loopback error (0.0% recovery), emphasizing the necessity of local context boundaries.

4.5 Theoretical Analysis: Semantic Density & Scale Degradation

To map fine-grained degradation and semantic crowding, we encode tool descriptions with 'all-MiniLM-L6-v2' and compute nearest-neighbor cosine similarities alongside empirical selection error and JSON schema compliance from $N = 5$ to $N = 200$.

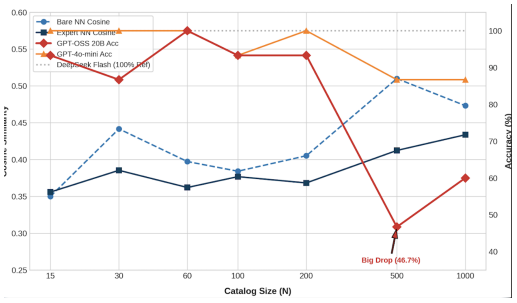

Figure 9: Semantic density vs. routing accuracy.

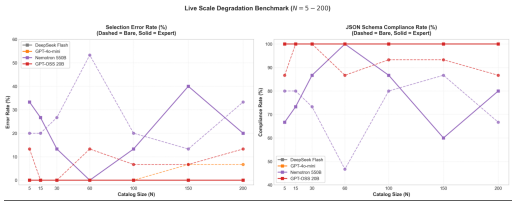

Figure 10: Attention degradation curve ($N = 5 \ldots 200$).

Table 9 and Figures 9–10 show that flagship models (DeepSeek V4 Flash and GPT-4o-mini) maintain 0.0% selection error up to $N = 100$, demonstrating that failure curves are non-linear and model-dependent.

5 Architectural Discussion

Our findings suggest a practical design pattern for autonomous agent architectures:

1. Decouple Routing from Computation: Use language models strictly to select tools and parameters, delegating numeric calculations to sandboxed execution environments.
2. Enforce Local Context Boundaries: Restrict per-stage tool catalogs ($N \leq 15$) using dynamic stage routers or state graphs to prevent attention degradation across long trajectories.

Table 9: Measured Scale Degradation & Schema Compliance (%: Bare / Expert).

Size (N)	Metric (%)	DeepSeek V4 Flash	GPT-4o-mini	Nemotron 550B	GPT-OSS 20B
$N = 5$	Selection Error	0.0 / 0.0	0.0 / 0.0	20.0 / 33.3	13.3 / 0.0
	Schema Compliance	100.0 / 100.0	100.0 / 100.0	80.0 / 66.7	86.7 / 100.0
$N = 15$	Selection Error	0.0 / 0.0	0.0 / 0.0	20.0 / 26.7	0.0 / 0.0
	Schema Compliance	100.0 / 100.0	100.0 / 100.0	80.0 / 73.3	100.0 / 100.0
$N = 30$	Selection Error	0.0 / 0.0	0.0 / 0.0	26.7 / 13.3	0.0 / 0.0
	Schema Compliance	100.0 / 100.0	100.0 / 100.0	73.3 / 86.7	100.0 / 100.0
$N = 60$	Selection Error	0.0 / 0.0	0.0 / 0.0	53.3 / 0.0	13.3 / 0.0
	Schema Compliance	100.0 / 100.0	100.0 / 100.0	46.7 / 100.0	86.7 / 100.0
$N = 100$	Selection Error	0.0 / 0.0	0.0 / 0.0	20.0 / 13.3	6.7 / 0.0
	Schema Compliance	100.0 / 100.0	100.0 / 100.0	80.0 / 86.7	93.3 / 100.0
$N = 150$	Selection Error	0.0 / 0.0	6.7 / 0.0	13.3 / 40.0	6.7 / 0.0
	Schema Compliance	100.0 / 100.0	100.0 / 100.0	86.7 / 60.0	93.3 / 100.0
$N = 200$	Selection Error	0.0 / 0.0	6.7 / 0.0	33.3 / 20.0	13.3 / 0.0
	Schema Compliance	100.0 / 100.0	100.0 / 100.0	66.7 / 80.0	86.7 / 100.0

3. Layer Guardrails for Recovery: Combine AST validation with bounded re-prompting to catch prompt injections and schema mismatches before executing side-effects.

6 Limitations and Future Work

While our benchmark suite spans 8 model families, primary empirical tasks are concentrated in quantitative finance and data science. Future work will extend this framework to open-ended web browsing and software engineering environments. Additionally, production deployments require hardened containerized sandboxes (e.g., gVisor, Docker) beyond local Python subprocesses.

7 Conclusion

This paper presented an empirical evaluation of LLM agent reliability across tool selection, code execution, guardrails, and multi-step orchestration. We demonstrated that expert process templates combined with sandboxed execution resolve atomic execution failures, while stage-wise tool partitioning achieves 93.3%–100.0% end-to-end success in live multi-step benchmarks.

Code and Data Availability

Benchmark datasets, evaluation scripts, process templates, and execution harnesses will be released as an anonymized open-source repository upon acceptance, to preserve double-blind review.

AI use statement

Generative AI tools were used solely for code formatting assistance and preliminary LaTeX layout verification. All research hypotheses, experimental designs, benchmark execution, mathematical formulations, and textual analyses were conducted independently by the authors.

Reproducibility statement

To ensure complete reproducibility, all dataset generation scripts, skill catalog definitions ($N = 5 \ldots 1000$), evaluation scripts, and prompt templates are archived in an anonymized

repository, made available to reviewers via the OpenReview submission system and to be de-anonymized and publicly released upon acceptance. Detailed hyperparameter configurations and model API versions are specified in Appendix A.1.

Ethics statement

This work adheres to the ICLR Code of Ethics. All model evaluations were conducted using public APIs or open-weight releases without human subjects, personal data collection, or privacy violations.

References

Anonymous. From multi-agent to single-agent: Compiling collaboration into pluggable skills. arXiv preprint arXiv:2601.04748, 2026.

Center for AI Safety and Scale AI. Humanity's last exam. Nature, 2025.

Minghao Li, Feifan Song, Bowen Yu, Haiyang Yu, Zhouyu Li, Jiaming Ji, An Yang, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Api-bank: A benchmark for tool-augmented llms. arXiv preprint arXiv:2304.08244, 2023.

Yujia Qin, Shengding Shi, Yining Ye, Luoshang Rong, Kai Shen, Zhenzhen Tang, Yuxiang Liang, Yankai Chen, Huadong Wang, Yuhui Huang, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. arXiv preprint arXiv:2307.16789, 2023.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. arXiv preprint arXiv:2302.04761, 2023.

A Appendix

A.1 Detailed Model Specifications

Table 10 lists the exact model checkpoints and temperature settings used across all benchmark phases.

Table 10: Model Checkpoints and API Provider Configurations.

Model	Provider / Version	Temperature	Max Tokens
DeepSeek V4 Flash	DeepSeek API	0.0	2048
DeepSeek V4 Pro	DeepSeek API	0.0	2048
GPT-4o-mini	OpenAI (2024-07-18)	0.0	2048
GPT-OSS 20B	OpenRouter / Local	0.0	2048
Gemini 3.1 Flash Lite	Google AI Studio	0.0	2048
Claude 4.5 Haiku	Anthropic API	0.0	2048
Nemotron-3-Ultra 550B	NVIDIA NIM	0.0	2048
Gemma-4 26B	Local vLLM	0.0	2048

A.2 AST Script Integrity Inspection Rules

The AST guardrail parses generated Python scripts into syntax trees and validates against the following constraints:

1. Syntax Validation: ‘ast.parse(code)‘ must succeed without raising ‘SyntaxError‘.
2. Disallowed System Calls: Nodes of type ‘ast.Call‘ invoking ‘os.system‘, ‘subprocess.Popen‘, ‘eval‘, or ‘exec‘ without explicit sandbox wrapping are flagged.
3. Return Schema Check: Scripts must define an explicit ‘main()‘ function returning a JSON-serializable dictionary containing the key ‘”status”‘.