Agent Seer: Synthesizing Scenarios from Specification Understanding

Anonymous ACL submission

Abstract

Evaluating AI agents that use external tools requires realistic test scenarios that capture how practitioners compose tools and iterate across conversation turns. Constructing such scenarios by hand demands deep domain expertise, does not scale across tool ecosystems, and produces static benchmarks that cannot track evolving APIs. We observe that tool specifications—function names, natural-language descriptions, and typed parameter schemas—already encode sufficient semantic information to synthesize realistic evaluation scenarios without manual curation or live tool execution. **Agent Seer** builds off this latent information: from a single Model Context Protocol (MCP) specification, with no examples, no live tool access, and no domain-specific tuning. This pipeline enriches raw schemas, generates graded scenarios with synthetic tool outputs, and expands them into mock-data-grounded multi-turn dialogues that exhibit strong tool-calling correctness and conversational coherence. Evaluation quality is measured by applying this pipeline on seven MCP specifications spanning diverse domains and tool-suite sizes and measuring the tool-calling correctness and conversational coherence. The pipeline achieves strong quality across all domains, with complete tool coverage on small and medium specifications. Two findings emerge within this analysis: parameter schema complexity is the strongest correlate of quality variation—tool-suite size plays a smaller, orthogonal role—and argument value accuracy is the dominant failure mode among imperfect scenarios, a sub-dimension invisible to coarse-grained name-match metrics.

1 Introduction

The deployment of large language model (LLM)-based agents capable of autonomously calling external tools is increasingly common in enterprise software environments. Agents backed by calendar APIs, project management systems, communications platforms, and internal databases are being used to automate workflows that previously required human intervention. Despite rapid progress in agent capabilities, the evaluation of such systems remains laborious and fragile.

Three problems limit the current state of agent evaluation:

The curation bottleneck. Realistic evaluation scenarios must connect user intent to specific tool calls, fill parameters with plausible values, and capture how tools chain across turns. Hand-authored benchmarks (Froger et al., 2026; Liu et al., 2023a; Yao et al., 2025) produce high-fidelity scenarios but their coverage is bounded by curation effort; scaling such efforts across the combinatorial space of tool pathways is impractical by hand.

The static benchmark problem. A fixed benchmark ceases to reflect reality as tool APIs evolve. An agent that scores highly on a snapshot benchmark may be evaluated against tool descriptions that no longer match the production environment.

The multi-turn evaluation gap. Conversational agents require evaluation across interaction sequences, not just single-turn prompts. Generating multi-turn scenarios where follow-up turns react to specific tool outputs—rather than repeating generic instructions—is particularly difficult without access to real tool responses.

Together these define the **cold-start evaluation problem**: producing realistic evaluation data for a tool suite that has none. The shortage is sharpest for new, private, or rapidly evolving APIs, and persists across the long tail of enterprise tool suites that wrap or extend publicly described systems. We address it by observing that tool specifications—function names, natural-language descriptions, and typed parameter schemas—already encode much of the semantic information needed to synthesize

evaluation data: enough for an LLM to infer plausible workflows, fill parameters with realistic values, synthesize what tool responses would look like, and construct multi-turn dialogues grounded in that synthetic data. The bottleneck shifts from *human curation* to *structured extraction*.

Agent Seer realizes this as a four-stage pipeline—semantic interpretation, scenario generation, mock output synthesis, and multi-turn expansion—that converts raw MCP tool specifications into self-contained evaluation harnesses. Each stage consumes validated structured outputs from the previous, so schema violations are caught at boundaries rather than propagating. The resulting harnesses are decoupled from any execution backend: any MCP-compatible framework can consume them to run and grade agents.

The primary contributions are:

1. A **spec-to-harness generation pipeline** that produces complete evaluation scenarios—expected tool sequences, calibrated mock tool outputs, and data-grounded multi-turn dialogues—from MCP tool specifications alone, without live tool execution or manual annotation.
2. A **structured harness artifact format** (Section 3.5) with held-out oracles and mock outputs, enabling any MCP-compatible evaluation framework to run agents against the generated scenarios as self-contained artifacts.
3. **Empirical characterization** of generation quality and tool coverage across seven MCP specifications spanning diverse enterprise domains, showing that quality variation is more strongly correlated with parameter schema complexity than with tool-suite size, and that complete coverage is achievable on small and medium specifications.
4. A **failure-mode characterization** identifying argument sub-dimension cascading as the dominant generation failure mechanism—a failure class invisible to coarse-grained name-match metrics—with domain-specific signatures characterizing where and why the pipeline falls short.

2 Background and Related Work

Evaluating tool-calling agents requires evaluation data—scenarios, expected tool sequences, and representative tool outputs. For established public APIs this can be hand-curated or mined from usage logs, but for new, private, or rapidly evolving tool suites no such data exists, and what does may not cover niche or challenging use cases. Even when curation is feasible, identifying the user goals that matter and connecting them to tool sequences that reflect how practitioners compose tools, handle partial results, and iterate across turns requires deep domain expertise. Agent Seer addresses this cold-start gap through *execution-free harness synthesis from a structured tool specification*.

2.1 Agent Benchmarks

Agent evaluation has progressed from single-function prediction (Qin et al., 2024; Patil et al., 2024) through multi-step tool use over curated API corpora (Qin et al., 2024; Lu et al., 2025; Chen et al., 2024) to multi-turn, policy-grounded evaluation with simulated users (Yao et al., 2025; Barres et al., 2025; Wang et al., 2024). General benchmarks such as GAIA (Froger et al., 2026), AgentBench (Liu et al., 2023a), and WorkArena (Drouin et al., 2024) provide rich evaluation environments, while recent MCP-centric suites—MCPVerse (Lei et al., 2025), MCP-AgentBench (Guo et al., 2026), Toolathlon (Li et al., 2026)—reflect the growing adoption of MCP (Anthropic, 2024) as a standard tool integration layer. All of these benchmarks are constructed through manual curation or require live tool access, and remain static once released, leaving the cold-start problem unaddressed.

2.2 Synthetic Data Generation

Prior work on generating synthetic data falls into three clusters, distinguished by their reliance on live tool execution.

Training trajectories from live execution. The largest cluster generates fine-tuning trajectories via real tool execution: APIGen (Liu et al., 2024; Prabhakar et al., 2025) uses execution-based verification, TOUCAN (Xu et al., 2025) scales to 1.5M trajectories from live MCP servers, GEM (Xu et al., 2026) mines trajectories from text corpora, and Agent World Model (Wang et al., 2026) synthesizes RL environments. Most require live tool invocation.

Simulated tool environments. A second cluster replaces live tools with simulated environments—for training: Simia (Li et al., 2025), Gecko (Zhang et al., 2026), LOGIGEN (Zeng et al., 2026); for evaluation: τ-bench (Yao et al., 2025), τ^2-bench (Barres et al., 2025), and ToolSandbox (Lu

et al., 2025). Concurrent work (Anonymous, 2026) establishes a formal world state from specifications and perturbs the observed state to measure robustness; the present work instead generates the evaluation scenarios themselves, and the resulting harnesses can feed such evaluators.

Spec-only generation. An emerging cluster generates synthetic data purely from specifications. DiGiT-TC (Crouse et al., 2026) back-translates tool-call sequences into user requests for fine-tuning data; FuncBenchGen (Maekawa et al., 2026) defines contamination-free task benchmarks via DAG-modelled call dependencies.

2.3 Evaluation Methodology

The LLM-as-judge paradigm is well established (Liu et al., 2023b; Kim et al., 2024; Es et al., 2024). For tool-use evaluation, prior work uses exact match (Qin et al., 2024), execution success rate (Qin et al., 2024), and single-call binary AST matching (Patil et al., 2025a). Agent GPA (Jia et al., 2026) decomposes traces into Goal-Plan-Action stages but evaluates on live task environments. FuncBenchGen (Maekawa et al., 2026) finds that models systematically propagate stale or incorrect arguments across chained calls despite syntactic validity—a multi-step failure mode that coarse-grained name-match metrics miss entirely, motivating the per-argument sub-dimension decomposition we apply here. We draw on this methodology to assess generation quality along two complementary dimensions (tool-calling correctness and conversational coherence), applying LLM-as-judge scoring with structured multi-dimensional prompts at each turn.

2.4 Positioning

Agent Seer takes only tool specifications as input (like spec-only systems) but uses an LLM to synthesize plausible tool responses rather than requiring a runtime environment (like simulated environments for evaluation). The contribution is not the prompt-driven generation mechanism—shared with APIGen, TOUCAN, and others—but the *structure imposed on it*: a four-stage pipeline with validated structured outputs at each boundary, producing harnesses decoupled from any execution backend. The quality assessment framework is shared infrastructure also employed by concurrent work (Anonymous, 2026). Table 17 in Appendix C situates these contributions against prior literature.

3 Pipeline Stages

The pipeline has four stages (schema flow in Figure 2, Appendix F). Each consumes validated structured outputs from the previous stage; schema constraints ensure malformed artifacts are caught at boundaries rather than propagating downstream.

3.1 Tool Interpretation

The first stage converts raw MCP tool specifications into semantically-enriched descriptions. For each tool, the module issues a structured prompt to an LLM requesting four semantic fields: a functional description, required parameters with semantic roles, primary use case, and organizational context (prompt templates in Appendix D; output schemas in Appendix F). This connects terse API documentation to richer scenarios.

3.2 Scenario Generation

The scenario generation module produces realistic enterprise workflow scenarios at two complexity tiers. **Simple scenarios** target everyday operational tasks: single-domain, short tool call chains. **Complex scenarios** target novel, multi-domain workflows combining tools in sophisticated ways. Both tiers are implemented through prompt engineering rather than structural constraints.

Each generated scenario includes a title, user-facing instruction, ordered list of expected tool calls with parameter values, novelty explanation, and a natural follow-up question. The output schema embeds **structured reasoning trace** fields: each tool call carries a *quick_explanation* justifying why the call is made, and each scenario includes a *novelty_reason* explaining its evaluation value. These fields force the LLM to reason about tool selection and workflow composition during generation.

3.3 Mock Output Generation

For each function call in the sequence, the module produces a synthetic tool output. The pipeline accepts an optional example outputs field, making it a spectrum: fully unsupervised from specifications alone, but able to incorporate available traces to improve fidelity. When examples are provided, the generator matches their structure; when absent, generation relies on the tool description alone. Each mock output carries a grounding tier (`high`/`medium`/`low`) recording the availability of reference material.

3.4 Multi-Turn Scenario Expansion

The multi-turn expansion stage takes a scenario and its mock outputs and emits a list of conversational turns (prompt template in Appendix D; output schema in Figure 2); the output is a list, but the prompt does not prescribe how many turns to produce.

Splits target natural phase boundaries so the resulting turns exercise two multi-turn tool-calling patterns formalized by BFCL v3 (Patil et al., 2025b): *multi-step* sequences, where each call depends on the output of the previous one, and *multi-hop* patterns, where independent calls gather information that must be synthesized. Splitting at phase boundaries (rather than at arbitrary points) preserves these dependency structures across turns. When the expansion yields only a single turn, it is discarded under the heuristic that the scenario lacked enough substance to split.

When the expansion succeeds, follow-up prompts reference concrete values from synthetic outputs—entity names, counts, status codes—rather than abstract task descriptions, producing **data-grounded** multi-turn dialogues.

3.5 Harness Artifact Format

The four stages produce a self-contained evaluation harness (Table 1). A downstream framework presents the prompt, feeds mock outputs as tool responses, and scores the agent's emitted calls against the scenario workflow as the held-out oracle—enabling evaluation on a previously unseen tool suite without live access.

Field	Stage	Content
`prompt`	2	Natural-language task goal
`expected_tools`	2	Ordered `AgentCall` objects (name + typed args)
`mock_outputs`	3	Synthetic JSON response per call with grounding tier
`conversation`	4	Multi-turn dialogue; each turn references mock output values
`oracle`	2	`expected_tools`, held out from the agent for scoring

Table 1: Fields of a generated evaluation harness.

4 Generation Quality Assessment

We assess generation quality using LLM-as-judge scoring (Liu et al., 2023b) along two complementary quality dimensions: tool-calling correctness (TC) and conversational coherence (Coh). This quality assessment framework is shared evaluation infrastructure also employed by concurrent work (Anonymous, 2026); the contribution of this paper is the generation pipeline, not the framework itself. The framework also supports oracle-grounded evaluation when reference data is available; we use the unsupervised path exclusively here, as the generated scenarios serve as both output and oracle.

4.1 Tool-Calling Scoring

The tool-calling evaluator independently scores four dimensions, decomposing the coarse-grained pass/fail signal used by most prior work (Qin et al., 2024) into orthogonal axes. An LLM judge scores each sub-dimension on 0–10; scores are normalized to 0–1 before aggregation.

Tool usage correctness captures necessity (Huang et al., 2023)—whether a tool was warranted at all—with overuse recorded as diagnostic but excluded from the aggregate, since necessity dominates appropriateness.

Tool selection correctness averages correctness, specificity, and completeness of the chosen tools.

Tool ordering correctness averages sequence logic, dependency handling, and execution efficiency. It is marked not applicable when only one tool is called, and is then excluded from the turn-level aggregate.

Tool argument correctness averages six sub-dimensions (completeness, name, value, type, format, relevancy). Cascading penalties are enforced through prompt instructions to the judge (Table 18, footnote): a wrong parameter name or missing required parameter zeros out value, type, and format; a wrong value cascades to type, format, and relevancy. A single critical error therefore collapses the argument mean.

The four dimensions combine via arithmetic mean per turn; conversation scores are the arithmetic mean of turn scores. Per-MCP rankings and the schema-complexity correlation are robust to harmonic-mean and minimum aggregation (Appendix B); the full prompt is in Appendix E.

4.2 Coherence Scoring

The coherence evaluator assesses five sub-aspects—logical flow, completeness, conciseness, topic relevance, and context retention—scored on a 1–3 scale, normalized to 0–1, and aggregated via arithmetic mean (full prompt in Appendix E).

5 Experimental Evaluation

We evaluate Agent Seer on seven publicly available MCP specifications, analyzing the quality, coverage, and failure modes of generated evaluation scenarios.

5.1 Experimental Setup

MCP specifications. Seven open-source MCP server specifications span diverse domains, tool counts, and schema complexity (Table 2). Sources: the official MCP reference server repository (Model Context Protocol, 2024) and the MCP server registry (Model Context Protocol, 2025).

MCP	Domain	Tools	$\bar{p}$	Schema
Illustrator	Creative	64	3.6	Nested obj
Selenium	Browser auto.	56	1.8	Flat state
Redis	Data store	47	2.1	Flat k-v
Git	Version ctrl.	33	11.2	Deep opt.
Elasticsearch	Search	20	1.8	Nested DSL
Slack	Communication	16	2.2	Mixed
Filesystem	File ops.	14	1.8	Flat

Table 2: MCP specifications used for evaluation. $\bar{p}$ = mean parameters per tool. **Schema** characterizes the dominant parameter structure: *Flat* = simple key-value or primitive parameters; *Nested obj/DSL* = parameters containing nested objects or domain-specific query languages; *Deep opt.* = deeply nested schemas with many optional fields; *Flat state* = flat parameters but stateful sequential semantics; *Mixed* = combination of flat and structured parameters across tools.

Generation model. All scenarios were generated using Gemini 2.5 Flash Lite with structured output mode enabled for schema-constrained generation at each pipeline stage. Pipeline stages used temperature 0.7. Structured-output validation failures triggered up to three retries before the record was discarded.

Evaluation method. Generated scenarios were scored using Gemini 2.5 Flash at temperature 0 (for deterministic scoring) as the LLM judge, following the procedure described in Section 4: (1) *Tool Calling*: LLM-as-judge scoring across four dimensions combined via arithmetic mean, with cascading penalties on argument sub-scores enforced through prompt instructions to the judge (scale 0–1); (2) *Coherence*: five sub-dimensions on a 1–3 raw scale, normalized to 0–1 and aggregated via arithmetic mean. The full corpus was additionally re-scored with an out-of-family judge (`Qwen3.5-122B-A10B-FP8`, Alibaba) to probe judge robustness; diagnostics appear in Appendix A.8.

Scale. The pipeline generated 337 scenarios across the seven MCPs (Table 6 in Appendix A), yielding 391 evaluation records. Multi-turn expansion succeeded for 54 scenarios (16.0% overall), heavily skewed toward complex scenarios (30.8% expansion rate vs. 2.8% for simple), as the expansion stage requires sufficient workflow substance to generate meaningful follow-up turns.

5.2 Quality Results

Overall quality. The pipeline achieves a mean unsupervised tool-calling score of 0.911 (95% bootstrap CI $[0.897, 0.925]$; median 0.979) and mean coherence of 0.855 (95% CI $[0.838, 0.872]$; median 0.933). 31.7% of records achieve a perfect tool-calling score, while only 2.3% score below 0.5. The distribution is concentrated in the upper range, reflecting consistent generation quality across specifications (see Figure 3 in Appendix G).

Cross-family co-evaluation. To check that these scores reflect the generated data rather than a single judge family, the corpus was re-scored with an out-of-family judge (`Qwen3.5-122B-A10B-FP8`, Alibaba; the primary judge is Google's Gemini 2.5 Flash). Tool-calling agrees at every grain: no mean shift ($\Delta\mu_{\text{TC}}\approx 0$, 95% CI $[-0.009, +0.008]$), record-level paired $r{=}0.79$ over $n{=}384$ paired records, per-MCP CIs overlap for all seven MCPs, and the MCP ranking is preserved ($\rho{=}0.86$). The failure-mode taxonomy replicates bilaterally: argument value-accuracy is the dominant sub-failure under both judges by a 4–5$\times$ margin (Gemini 240 records, Qwen35 252; Table 14). The judges diverge on the absolute level of coherence ($\Delta\mu_{\text{Coh}}\approx -0.16$, paired $r{=}0.42$). Coherence levels are therefore reported as judge-dependent; MCP-level coherence rank-preservation is partial ($\rho{=}0.46$), and the worst-coherence MCP differs across judges (Git under Gemini, Illustrator under Qwen 3.5). Full diagnostics appear in Appendix A.8.

Quality by MCP specification. Table 3 reports per-MCP scores.

Tool count and parameter schema complexity play distinct, orthogonal roles in quality variation across this sample. At the per-MCP grain ($n = 7$), tool count correlates positively but modestly with mean unsupervised tool-calling

MCP	n	TC [95% CI]	σ	Coh [95% CI]	Simp.	$\bar{w}$
Illustrator (64t)	36	0.898 [.85,.94]	0.131	0.855 [.80,.91]	0.934	3.3
Selenium (56t)	47	0.935 [.90,.96]	0.095	0.850 [.80,.90]	0.932	9.7
Redis (47t)	98	**0.966** [.95,.98]	0.089	0.902 [.87,.93]	0.986	1.3
Git (33t)	85	0.857 [.82,.90]	0.186	0.757 [.71,.80]	0.910	2.2
Elasticsearch (20t)	49	0.930 [.90,.96]	0.108	0.902 [.86,.94]	0.987	1.9
Slack (16t)	35	0.886 [.84,.93]	0.135	**0.938** [.91,.97]	0.934	1.4
Filesystem (14t)	41	0.876 [.82,.92]	0.163	0.825 [.77,.87]	0.925	2.0

Table 3: Unsupervised scores by MCP. TC = tool calling, Coh = coherence, Simp. = simple scenario TC, $\bar{w}$ = mean workflow length (tools per scenario). 95% intervals are percentile bootstrap ($B = 10{,}000$). All seven MCPs exceed 0.85 overall; all exceed 0.91 on simple scenarios.

($r = +0.40$), while parameter schema complexity correlates negatively—average parameters per tool ($r = -0.60$) and optional parameter fraction ($r = -0.66$). The two effects operate on different axes of an MCP and do not cancel: Selenium (56 tools) scores 0.935 while Filesystem (14 tools) scores 0.876, but Git (33 tools, 11.2 average parameters) is the lowest at 0.857.

To strengthen the inferential basis of these per-MCP observations, we additionally disaggregate to the tool level. Across the 222 unique in-spec tools appearing in any generated scenario, parameter count and optional-fraction both correlate negatively with mean unsupervised TC ($r = -0.29$ and -0.30, both $p < 0.001$), confirming the per-MCP direction at a substantially larger sample. The same schema features also correlate negatively with mean coherence at the tool level ($r = -0.41$ and -0.34, both $p < 0.001$). This second finding—not detectable at the per-MCP grain—suggests complex parameters degrade not only tool-calling correctness but also the agent's ability to communicate cleanly around those tools.

Coherence tells a different story: Slack achieves the highest coherence (0.938) because its scenarios follow structured messaging patterns that produce natural conversational flow, while Git has the lowest (0.757) because version control workflows involve complex multi-step operations with technical context. The two dimensions remain weakly correlated at the unit of evaluation: record-level $r = +0.23$ ($n = 381$) and tool-level $r = +0.16$ ($n = 222$, $p = 0.016$). The cross-MCP correlation between MCP means is higher ($r = +0.57$, $n = 7$) but reflects aggregation effects rather than a stronger underlying relationship. TC and coherence therefore provide largely independent diagnostic signals across grains.

Dimension	Perfect	Partial	Zero
Usage	98%	1%	1%
Selection	77%	19%	4%
Ordering†	71%	19%	10%
Arguments	42%	57%	1%

Table 4: Tool-calling dimension failure rates. †Ordering is evaluated only when multiple tools are called ($n = 181$).

Complexity breakdown. Complex scenarios degrade by -7.3pp relative to simple ones in tool calling (0.949 [95% CI 0.932, 0.965] $\rightarrow$ 0.877 [0.855, 0.897]) and -5.3pp in coherence (0.883 [0.857, 0.908] $\rightarrow$ 0.830 [0.807, 0.853]); the simple/complex CIs are non-overlapping in both dimensions. The effect varies by domain (Table 7 in Appendix A): Elasticsearch shows the largest degradation (-12.2pp), followed by Git (-11.1pp, from 0.910 to 0.799). Selenium is stable (0.932$\rightarrow$0.935), suggesting that its long sequential workflows are no harder to compose at higher complexity.

5.3 Failure Mode Analysis

Tool-calling dimension failures. Table 4 reports per-dimension failure rates. Usage is near-perfect; selection is correct on 77% of records (4% zeros, reflecting the cost of choosing among semantically similar tools); ordering fails on 10% of multi-tool scenarios. **Argument correctness is the dominant challenge**: only 42% of records score perfectly and 57% score partial, driven by value-accuracy errors that degrade scores without collapsing them.

Argument failure patterns. *Value accuracy* dominates argument failures (223 records), followed by *relevancy* (44), *format* (35), *type* (31), *completeness* (16), and *name accuracy* (11). Counts attribute each failing record to its lowest-scored sub-dimension (single assignment); the judge-comparison appendix (Table 14) uses the looser criterion of any sub-dimension scoring below 1.0, which yields larger per-sub-dimension totals. The pipeline reliably generates correct parameter names and types but struggles with precise values, particularly for optional parameters with ambiguous semantics.

Failure concentration by MCP. Failures are sparse and unevenly distributed: Git shows the highest record-level failure rate, followed by Filesystem and Redis; the other four MCPs are near-zero. Per-domain breakdowns and

aggregation-rule sensitivity appear in Appendices A.6 and B.

Illustrative failure: Redis argument ambiguity. A representative pattern occurs in Redis, where several commands carry optional parameters that the pipeline omits when context implies them. The `set` tool, for example, accepts an optional `expiration` field that is frequently skipped when the scenario implies time-bounded storage. The pipeline consistently generates the correct function name, key, and value arguments but omits this expiry parameter, which the argument scoring framework flags as a *completeness* sub-dimension failure; a coarse name-match metric would score these records as fully correct. This pattern—correct tool selection with subtly incomplete argument specification—is the primary driver of Redis failures and illustrates why argument correctness must be decomposed below the function-call level: without the sub-dimension breakdown, the dominant failure mode in this domain is invisible.

Illustrative failures: Git's two mechanisms. Git failures decompose into two distinct mechanisms operating at different scenario complexities. The first, observable on simple scenarios, is *tool-name hallucination*: in three Git scenarios the pipeline emits real Git CLI commands that are not in the MCP specification (`fetch`, `revert`, `filter-repo`), a pretraining-knowledge leak past the spec. These three records account for half of Git's six records with TC$<$0.5 and for the gap between Git's simple-scenario TC (0.910) and the other six MCPs (0.93–0.99 simple). Across the full corpus, hallucinated calls are 0.336% of all tool invocations (3 of 893), entirely concentrated in Git.

The second mechanism, observable on complex scenarios, is *parameter overload*—localized to specific high-parameter tools rather than the full MCP. Git tools average 11.2 parameters (3$\times$ the next highest), 95% optional, and the `ref` parameter appears in seven tools with different semantics ("commits starting from" in `log`, "compare against" in `diff`, "show object at" in `show`). The pipeline selects the right tool (selection score 0.802) but generates incorrect parameter values, producing a mean argument score of 0.780—the lowest of any MCP. The bottom-fifteen tools by unsupervised TC across the corpus include five Git tools (`blame`, `add`, `commit`, `log`, `show`), all high in parameter density. Complex Git scenarios degrade further (arguments 0.726) as multi-tool workflows compound per-call parameter errors.

MCP	Spec	Used	Cov.	Gini
Illustrator	64	36	56%	0.331
Selenium	56	56	100%	0.676
Redis	47	47	100%	0.146
Git	33	33	100%	0.340
Elasticsearch	20	20	100%	0.198
Slack	16	16	100%	0.177
Filesystem	14	14	100%	0.294

Table 5: Tool coverage and usage uniformity. Gini coefficient measures inequality (lower = more uniform).

Coherence taxonomy and cross-dimension pattern. The most frequent coherence issues are *missing information or shallow response* (373 records), *off-topic* (130), *non-sequitur* (86), and *self-contradiction or hallucination* (85); the full taxonomy is in Table 10. Record-level TC and Coh remain weakly correlated (r=0.23), and the "correct tools, poor coherence" quadrant (23% of records at a 0.75 threshold) spans all MCPs rather than concentrating in one domain—coherence shortfalls are a general pipeline property.

5.4 Domain Cluster Analysis

Grouping MCPs by target system type lets structural differences within each cluster act as a natural experiment. *Data Store* (Redis, Elasticsearch) pairs near-identical profiles and yields near-identical quality (Δ0.036 TC). *Developer/File* (Filesystem, Git) spans a 6$\times$ parameter-density gap (1.8 vs. 11.2) but only Δ0.019 in TC, with the cost localized to argument correctness (Git 0.780, Filesystem 0.894) while strong selection and ordering compensate. *Application/UI* (Illustrator, Selenium) is the most semantically distant pair: Selenium (0.935 TC, 56 tools) outperforms Illustrator (0.898 TC, 64 tools) because its low parameter density enables reliable argument generation in long sequential chains.

5.5 Coverage and Diversity

Tool coverage. Table 5 reports the fraction of available tools exercised. Within the small-to-medium range (14–56 tools), every tool in Redis, Selenium, Git, Elasticsearch, Slack, and Filesystem appears in at least one generated scenario. The only specification beyond that range, Illustrator (64 tools), reaches 56%—the only evidence of a coverage ceiling in this experiment.

Usage uniformity. The Gini coefficient over each MCP's tool-usage frequency distribution ranges from 0.146 (Redis) to 0.340 (Git)—near-uniform sampling. Selenium is the outlier (0.676), with long sequential workflows (mean 9.7 calls) concentrating usage on core navigation and interaction tools. Across all MCPs, the pipeline yields 781 unique co-occurrence pairs from 138 multi-tool scenarios and 108 unique scenario categories (Appendix A.5).

5.6 Summary of Findings

Across 337 scenarios on seven MCPs, the pipeline achieves mean TC 0.911 and mean coherence 0.855 with complete tool coverage on small and medium specifications. The principal observations:

1. **Parameter schema complexity is the strongest correlate of quality variation in this sample**; tool-suite size plays a smaller, orthogonal role. Schema correlations are negative at both grains (per-MCP $r = -0.60/-0.66$ on TC; tool-level $r = -0.29/-0.30$ on TC and $-0.41/-0.34$ on coherence, $p < 0.001$ throughout); tool count correlates positively but modestly with TC ($r = +0.40$ per-MCP).
2. All seven MCPs exceed 0.91 on simple scenarios—including Git despite 11.2 avg params per tool—and complex scenarios degrade by 7.3pp on average. Domain clusters confirm the pattern: same-profile pairs yield similar TC; differing-profile pairs stay close when parameter density dominates.
3. TC and coherence provide largely independent diagnostic signals (record-level $r = +0.23$, tool-level $r = +0.16$), with characteristic inversions (Slack high coherence, low TC; Selenium best ordering).
4. Argument correctness is the primary challenge: 57% of records score partial on arguments, value-accuracy dominant; on Git, failures decompose into pretraining-leak tool-name hallucination on simple scenarios and parameter-overload localized to a handful of high-parameter tools on complex scenarios.
5. Headline tool-calling and the dominant argument-value-accuracy failure mode are robust to an out-of-family judge swap (paired $r{\approx}0.79$ on TC, MCP rank $\rho{=}0.86$); absolute coherence levels and MCP-level coherence rank-preservation are judge-dependent (Appendix A.8).

6 Conclusion

We have presented Agent Seer, a four-stage pipeline that converts MCP tool specifications into complete evaluation harnesses—graded scenarios, mock tool outputs, and multi-turn dialogues—without live tool execution or manual annotation. Across seven structurally diverse specifications, the pipeline reaches full tool coverage on small and medium MCPs and surfaces consistent diagnostic patterns: parameter schema complexity is the strongest correlate of quality variation in this sample, and argument value accuracy is the dominant remaining sub-failure. These findings are grounded in $n = 7$ specifications and should be read as observations within the experiment rather than universal claims; the durable contribution is the methodology, which produces harnesses as reusable artifacts that feed deterministic executors (Anonymous, 2026), live tool environments (Yao et al., 2025), or simulated agents (Li et al., 2025)—closing the cold-start evaluation gap for MCP-compatible tool suites.

Limitations

Ground truth reliability. The most significant limitation is reliance on LLM-generated ground truth, which introduces systematic biases from the generating model. The framework is best understood as a proxy evaluation tool that identifies broad capability gaps and relative performance differences.

Cross-call referential integrity. Mock outputs for sequential calls are currently generated independently, meaning IDs or values may not align across dependent calls. A shared state dictionary across a workflow would address this.

Coverage ceiling. At 64 tools (Illustrator), coverage drops to 56%. Targeted generation strategies—such as iterative generation with coverage-aware tool sampling—would be needed for larger specifications.

Specification scope. Cross-domain harness generation and analysis spanning multiple MCP specifications in a single workflow is structurally straightforward within the existing pipeline as well, but is outside the scope of this evaluation and remains a rich direction for future work. Extension to other spec formats (OpenAPI, gRPC, function-calling schemas) is similarly structurally straightforward—the same name + typed-parameter scaffolding is present—and is a natural next step for evaluating the framework's generality.

Complexity stratification. The simple/complex

distinction uses prompt framing rather than structural enforcement. Post-generation filters based on structural complexity metrics would improve stratification.

Experimental scope. Seven MCP specifications and a single generation model (Gemini 2.5 Flash Lite) cannot establish universal claims. Multi-turn evaluation records are limited ($n = 54$), with expansion heavily skewed toward complex scenarios (30.8% expansion rate) versus simple scenarios (2.8%), reflecting the stage's dependence on sufficient workflow substance to generate meaningful follow-up turns. This constrains statistical power for multi-turn findings.

LLM-as-judge circularity. Both generation and quality verification rely on LLMs, raising the concern that scenarios that "look good to an LLM" score well regardless of actual quality. We address this in two ways. First, an evaluator–generator capability gap: the judge (Gemini 2.5 Flash) has surplus capacity over the weaker generator (Gemini 2.5 Flash Lite), consistent with teacher–student evaluation paradigms; empirically, Section 5 shows the judge discriminates meaningful, domain-specific failure patterns (argument cascading, semantic ambiguity in Redis) rather than producing uniformly high scores. Second, an out-of-family replication: re-scoring the full 391-record corpus with `Qwen3.5-122B-A10B-FP8` (Alibaba family) yields paired Pearson $r{\approx}0.79$ on tool-calling ($n{=}384$) with no mean shift, preserves the MCP ranking ($\rho{=}0.86$), and reproduces the dominant argument value-accuracy failure mode bilaterally (Gemini 240 records, Qwen 3.5 252; 4–5$\times$ margin over the next sub-dim under both judges). Coherence shows a systematic stricter-judge shift ($\Delta\mu{\approx}-0.16$) with moderate record-level correlation ($r{=}0.42$); absolute coherence levels and MCP-level coherence rank-preservation should therefore be read as judge-dependent. Full agreement tables, per-MCP bootstrap CIs under both judges, and the Bland–Altman analysis appear in Appendix A.8. A systematic human evaluation study correlating framework scores with human quality judgments remains an important direction for future work.

References

Anonymous. 2026. A MetaWorld framework for cold-start evaluation of tool-calling agents. Under review.

Anthropic. 2024. Model context protocol. https://modelcontextprotocol.io.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. τ^2-bench: Evaluating conversational agents in a dual-control environment. *Preprint*, arXiv:2506.07982.

Tommaso Castellani, Naimeng Ye, Daksh Mittal, Thomson Yen, and Hongseok Namkoong. 2025. SynthTools: A framework for scaling synthetic tools for agent development. *Preprint*, arXiv:2511.09572.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. 2024. T-eval: Evaluating the tool utilization capability of large language models step by step. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9510–9529, Bangkok, Thailand. Association for Computational Linguistics.

Maxwell Crouse, Ibrahim Abdelaziz, Kshitij Fadnis, Siva Sankalp Patel, Kinjal Basu, Chulaka Gunasekara, Sadhana Kumaravel, Asim Munawar, and Pavan Kapanipathi. 2026. Simulating complex multi-turn tool calling interactions in stateless execution environments. *Preprint*, arXiv:2601.19914.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. 2024. WorkArena: How capable are web agents at solving common knowledge work tasks? In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 11642–11662. PMLR.

Shahul Es, Jithin James, Luis Espinosa-Anke, and Steven Schockaert. 2024. RAGAS: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th European Chapter of the Association for Computational Linguistics (System Demonstrations)*. ArXiv:2309.15217.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, and 5 others. 2026. Gaia2: Benchmarking LLM agents on dynamic and asynchronous environments. In *Proceedings of the Fourteenth International Conference on Learning Representations*.

Zikang Guo, Benfeng Xu, Chiwei Zhu, Wentao Hong, Xiaorui Wang, and Zhendong Mao. 2026. MCP-AgentBench: Evaluating real-world language agent performance with MCP-mediated tools. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 30888–30896.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan,

Neil Zhenqiang Gong, and Lichao Sun. 2023. MetaTool benchmark for large language models: Deciding whether to use tools and which to use. *Preprint*, arXiv:2310.03128.

Allison Sihan Jia, Daniel Huang, Nikhil Vytla, Seung Won Wilson Yoo, Nirvika Choudhury, Shayak Sen, John C. Mitchell, and Anupam Datta. 2026. What is your agent's gpa? a framework for evaluating agent goal-plan-action alignment. *arXiv:2510.08847*.

Seungone Kim, Juyoung Suk, Shayne Longpre, Bill Yuchen Lin, Jamin Shin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. 2024. Prometheus 2: An open source language model specialized in evaluating other language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 4334–4353, Miami, Florida, USA. Association for Computational Linguistics.

Fei Lei, Yibo Yang, Wenxiu Sun, and Dahua Lin. 2025. MCPVerse: An expansive, real-world benchmark for agentic tool use. *Preprint*, arXiv:2508.16260.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, Junteng Liu, Zhaochen Su, Yiyang Guo, Fan Zhou, Lueyang Zhang, Juan Michelini, Xingyao Wang, Xiang Yue, Shuyan Zhou, and 2 others. 2026. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. *arXiv:2510.25726*.

Yuetai Li, Huseyin A Inan, Xiang Yue, Wei-Ning Chen, Lukas Wutschitz, Janardhan Kulkarni, Radha Poovendran, Robert Sim, and Saravan Rajmohan. 2025. Simulating environments with reasoning models for agent training. *Preprint*, arXiv:2511.01824.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2023a. AgentBench: Evaluating LLMs as agents. *Preprint*, arXiv:2308.03688.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023b. G-eval: NLG evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore. Association for Computational Linguistics.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. 2024. APIGen: Automated pipeline for generating verifiable and diverse function-calling datasets. In *Advances in Neural Information Processing Systems*, volume 37, pages 54463–54482.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping 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, Albuquerque, New Mexico. Association for Computational Linguistics.

Seiji Maekawa, Jackson Hassell, Pouya Pezeshkpour, Tom Mitchell, and Estevam Hruschka. 2026. Towards reliable benchmarking: A contamination free, controllable evaluation framework for multi-step LLM function calling. In *The Fourteenth International Conference on Learning Representations*.

Model Context Protocol. 2024. Reference MCP servers. GitHub, `https://github.com/modelcontextprotocol/servers`. Official reference and example MCP server implementations. Accessed: 2026-03-27.

Model Context Protocol. 2025. MCP server registry. `https://registry.modelcontextprotocol.io`. Official registry of MCP servers. Accessed: 2026-03-27.

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

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

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems*.

Akshara Prabhakar, Zuxin Liu, Weiran Yao, Jianguo Zhang, Ming Zhu, Shiyu Wang, Zhiwei Liu, Tulika Awalgaonkar, Haolin Chen, Thai Hoang, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. 2025. APIGen-MT: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. In *Advances in Neural Information Processing Systems*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Zhenzhen Ren, Xinpeng Zhang, Zhenxing Qian, Yan Gao, Yu Shi, Shuxin Zheng, and Jiyan He. 2025. GTM: Simulating the world of tools for AI agents. *arXiv:2512.04535*.

Abhigya Verma, Seganrasan Subramanian, Nandhakumar Kandasamy, and Naman Gupta. 2025. FABRIC: Framework for agent-based realistic intelligence creation. *Preprint*, arXiv:2510.17995.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2024. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. In *Proceedings of the Twelfth International Conference on Learning Representations*.

Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. 2026. Agent world model: Infinity synthetic environments for agentic reinforcement learning. *Preprint*, arXiv:2602.10090.

Zhangchen Xu, Adriana Meza Soria, Shawn Tan, Anurag Roy, Ashish Sunil Agrawal, Radha Poovendran, and Rameswar Panda. 2025. Toucan: Synthesizing 1.5m tool-agentic data from real-world mcp environments. *Preprint*, arXiv:2510.01179.

Zhihao Xu, Rumei Li, Jiahuan Li, Rongxiang Weng, Jingang Wang, Xunliang Cai, and Xiting Wang. 2026. Unlocking implicit experience: Synthesizing tool-use trajectories from text. *arXiv:2601.10355*.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2025. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *Proceedings of the Thirteenth International Conference on Learning Representations*.

Yucheng Zeng, Weipeng Lu, Linyun Liu, Shupeng Li, Zitian Qu, Chenghao Zhu, Shaofei Li, Zhengdong Tan, Mengyue Liu, Haotian Zhao, Zhe Zhou, and Jianmin Wu. 2026. Logigen: Logic-driven generation of verifiable agentic tasks. *arXiv:2603.00540*.

Zeyu Zhang, Guohao Li, Zhenchang Xing, Alexandros Apostolopoulos, Yu Lin Lee, and Liang Zheng. 2026. Gecko: A simulation environment with stateful feedback for refining agent tool calls. *arXiv:2602.19218*.

A Extended Experimental Results

A.1 Evaluation Scale

Table 6 breaks down the 337 generated scenarios and 391 evaluation records by MCP specification. Records exceed scenarios because multi-turn scenarios contribute one record per turn (single-turn entry plus one multi-turn entry per additional turn).

	Scenarios	Records
Illustrator	27	36
Selenium	34	47
Redis	96	98
Git	72	85
Elasticsearch	42	49
Slack	33	35
Filesystem	33	41
Total	337	391

Table 6: Evaluation scale by MCP specification.

A.2 Complexity Breakdown

Table 7 reports mean tool-calling scores split by complexity tier (simple vs. complex) for each MCP. The Δ column shows the score change from simple to complex; a negative value indicates quality degradation under higher complexity. Elasticsearch shows the largest degradation (-12.2pp), followed by Git (-11.1pp). Selenium is notably stable, scoring slightly higher on complex scenarios.

MCP	Simple	Complex	Δ
Illustrator	0.934	0.865	-0.069
Selenium	0.932	0.935	$+0.003$
Redis	0.986	0.943	-0.043
Git	0.910	0.799	-0.111
Elasticsearch	0.987	0.865	-0.122
Slack	0.934	0.840	-0.094
Filesystem	0.925	0.834	-0.091

Table 7: Tool-calling scores by complexity tier and MCP.

A.3 Coherence Sub-Dimensions

Table 8 reports mean scores for each of the five coherence sub-dimensions on the raw 1–3 scale used by the judge. Conciseness is near-ceiling across all domains (mean 2.90, std 0.33), indicating the pipeline reliably produces appropriately scoped responses without unnecessary elaboration. Completeness is the primary bottleneck (mean 2.04), driven by responses that address the user's request but omit contextual detail that a practitioner would expect—the most actionable target for future pipeline improvements.

Sub-dimension	Mean	Std.
Conciseness	2.90	0.33
Context retention	2.78	0.59
Logical flow	2.64	0.75
Topic relevance	2.47	0.85
Completeness	2.04	0.93

Table 8: Coherence sub-dimension scores (raw 1–3 scale).

A.4 Grounding Tiers

Table 9 defines the grounding tiers assigned to each mock output at Stage 3. The tier records how much reference material was available to the LLM when generating the synthetic tool response: `high` when concrete example outputs were present in the tool specification, `medium` when similar tools provided transferable examples, and `low` when generation relied on the parameter schema alone. In the current experiments, all seven MCP specifications lack example outputs, so all 871 mock calls are annotated as `low`. The annotation is carried through to the harness artifact for downstream filtering when specs with richer documentation are used.

Tier	Condition
`high`	Concrete example outputs exist
`medium`	Examples from similar functions
`low`	No examples available

Table 9: Grounding tiers for mock output generation.

A.5 Workflow Composition and Category Diversity

Selenium generates the longest workflows (mean 9.71 calls, max 17) reflecting the sequential nature of browser automation tasks. Illustrator workflows are also long (mean 3.33, max 12), while Redis (mean 1.27) and Slack (mean 1.39) tend toward atomic single-tool scenarios. Category count (108 unique categories across 337 scenarios; 3.1 scenarios per category) scales with scenario count rather than tool count.

A.6 Failure Details

Failure concentration by MCP. Git exhibits the highest failure rate: 7% of records score below 0.5, followed by Filesystem (5%) and Redis (1%); four MCPs have zero failures. Git's failures are driven by argument correctness for tools with complex parameter schemas.

Argument failure patterns. Among argument failures, *value accuracy* is the dominant sub-dimension (223 records), followed by *relevancy* (44), *format compliance* (35), *type compliance* (31), *completeness* (16), and *name accuracy* (11).

Issue Group	Count
Missing info / shallow response	373
Off-topic or wrong focus	130
Non-sequitur or topic shift	86
Self-contradiction / hallucination	85
Ignores user constraints	54
No shared context (eval artifact)	44
Unnecessary elaboration	35
Lost thread / forgot prior info	13

Table 10: Coherence issue taxonomy grouped from evaluator notes.

A.7 Per-MCP Dimension Scores

MCP	Usage	Select	Order	Args
Illustrator	0.996	0.835	0.800	0.872
Selenium	0.965	0.891	0.989	0.890
Redis	1.000	0.954	0.823	0.952
Git	0.988	0.802	0.761	0.780
Elasticsearch	1.000	0.868	0.790	0.907
Slack	1.000	0.867	0.576	0.865
Filesystem	1.000	0.791	0.659	0.894

Table 11: Mean TC dimension scores by MCP (overall). Selenium's ordering (0.989) is the highest of any MCP, despite having the longest workflows. Git's arguments (0.780) are weakest, consistent with its high parameter density.

A.8 Out-of-Family Judge Replication

To assess whether reported scores are artifacts of within-family judge circularity, the full 391-record evaluation corpus was re-scored with an out-of-family judge: Qwen3.5-122B-A10B-FP8 (Alibaba family; vLLM-hosted, FP8 quantization). Each record's existing chat transcript was replayed against the alternate judge with only the evaluator model swapped; the generation pipeline and rubric are unchanged. After joining on the composite key (MCP, `conversation_id`), 384 records have paired tool-calling scores and 380 have paired coherence scores under both judges; all paired statistics below are computed on these sets.

Per-MCP means with bootstrap CIs. Table 12 reports per-MCP means and 95% percentile bootstrap CIs (B=2,000) under both judges. Tool-calling CIs overlap for every MCP. Coherence CIs are disjoint for all seven MCPs (the Qwen 3.5 judge is systematically lower), and the identity of the worst-coherence MCP differs across judges (Git under Gemini, Illustrator under Qwen 3.5).

MCP	n	TC Gem.	TC Qwen	Coh Gem.	Coh Qwen
Redis	98	0.966	0.964	0.902	0.766
Elasticsearch	49	0.930	0.935	0.902	0.744
Illustrator	36	0.898	0.821	0.855	0.604
Slack	35	0.886	0.884	0.938	0.707
Filesystem	41	0.876	0.871	0.825	0.701
Git	85	0.857	0.858	0.757	0.647
Selenium	47	0.935	0.919	0.850	0.655

Table 12: Per-MCP means under the Gemini judge (`gemini-2.5-flash`) and the Qwen 3.5 judge (`Qwen3.5-122B-A10B-FP8`). n counts come from the (MCP, `conversation_id`) join, which preserves all records; Gemini means match those reported in Table 3. 95% bootstrap CIs (omitted for space) overlap for every MCP on TC; coherence CIs are disjoint for all 7 MCPs. The bottom-coherence MCP differs across judges (Git under Gemini, Illustrator under Qwen 3.5).

MCP-level Spearman rank correlations across the seven MCPs are ρ=0.86 for tool-calling and ρ=0.46 for coherence. The top tool-calling MCP (Redis) is preserved under both judges; the bottom-coherence MCP differs (Git under Gemini, Illustrator under Qwen 3.5).

Record-level agreement. Table 13 reports paired-record agreement on the aggregate per-aspect unsupervised score (`agg_unsupervised_score`, the metric reported throughout the paper). Tool-calling agreement is strong; coherence is moderate.

Aspect	n	Pearson r	Spearman ρ	MAE	$\pm$0.1
Tool-calling	384	0.790	0.844	0.046	84%
Coherence	380	0.420	0.438	0.180	34%

Table 13: Record-level agreement between judges, computed on records keyed by (MCP, `conversation_id`) to avoid scenario-id collisions across MCPs. Bootstrap 95% CIs (B=2,000) omitted for space. MAE and the $\pm$0.1 agreement rate reproduce the within-tolerance qualitative pattern of the original analysis.

Systematic shift. The mean signed difference (Qwen 3.5 − Gemini) is $\Delta\mu_{\text{TC}}\approx 0$ (95% CI $[-0.009, +0.008]$; balanced sign-test) and $\Delta\mu_{\text{Coh}}\approx -0.16$ (95% CI $[-0.173, -0.138]$; the majority of non-tied records score lower under the Qwen 3.5 judge). Tool-calling shows no systematic bias; coherence is systematically stricter under the out-of-family judge. Without a third judge or a human anchor, neither judge can be designated "correct" on coherence; absolute coherence levels are therefore reported as judge-dependent.

Failure-mode taxonomy replication. Table 14 reports record counts where each argument sub-dimension scored below 1.0 under each judge. Both judges identify *value_accuracy* as the dominant argument failure mode by a 4–5× margin over the next-highest sub-dimension, replicating the central qualitative finding of Section 5; counts are within 5% across judges.

Sub-dimension	Gemini	Qwen 3.5
value_accuracy	**240**	**252**
relevancy	53	54
format_compliance	46	58
type_compliance	46	42
completeness	23	33
name_accuracy	16	21

Table 14: Argument sub-dimension failure counts (records with score < 1.0) under each judge. Value-accuracy dominance is preserved bilaterally; counts agree within 5% across judges.

Visual diagnostics. Figure 1 shows scatter (top row) and Bland–Altman (bottom row) plots for tool-calling and coherence. The TC scatter clusters tightly along $y{=}x$ and the TC Bland–Altman limits-of-agreement bracket zero symmetrically; the coherence scatter shows a visible offset below $y{=}x$ and the coherence Bland–Altman LoA is shifted ≈ -0.14 below zero with wider spread.

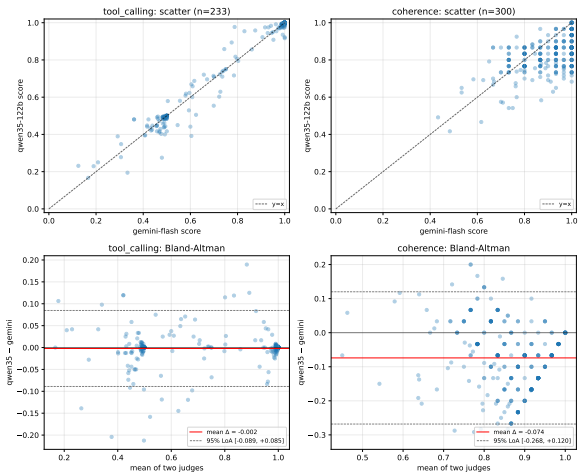


Figure 1: Record-level agreement between the Gemini judge (`gemini-2.5-flash`) and the Qwen 3.5 judge (`Qwen3.5-122B-A10B-FP8`). Top: scatter (Qwen 3.5 vs. Gemini); the TC cloud (left) lies on $y{=}x$, the coherence cloud (right) is offset below. Bottom: Bland–Altman; TC differences are centered on zero, coherence differences are shifted ≈ -0.138.

Interpretation. Per-MCP TC means, the MCP TC ranking, and the dominant argument-value-accuracy failure mode are robust to the judge

swap. Absolute coherence levels and MCP-level coherence rank-preservation (ρ_{Coh}=0.46) are judge-dependent. The worst-coherence MCP differs across judges (Git under Gemini, Illustrator under Qwen 3.5), reflecting the broader judge-dependence of coherence scoring rather than a contradiction in the underlying data.

MCP	$\lvert T\rvert$	$\bar{p}$	$p_{\max}$	Opt%	$\bar{w}$	TC	Coh
Illustrator	64	3.6	12	74	3.3	0.898	0.855
Selenium	56	1.8	5	26	9.7	0.935	0.850
Redis	47	2.1	6	34	1.3	0.966	0.902
Git	33	11.2	19	95	2.2	0.857	0.757
Elasticsearch	20	1.8	8	33	1.9	0.930	0.902
Slack	16	2.2	6	51	1.4	0.886	0.938
Filesystem	14	1.8	3	32	2.0	0.876	0.825

Table 15: Schema complexity profile. $|T|$ = tools, $\bar{p}$ = mean params/tool, $p_{\max}$ = max params, Opt% = optional parameter ratio, $\bar{w}$ = mean workflow length. Per-MCP correlations with mean TC: average parameters per tool $r = -0.60$, optional ratio $r = -0.66$, tool count $r = +0.40$ (positive but smaller). Tool-level disaggregation ($n = 222$) confirms the schema-complexity direction with $p < 0.001$: average parameters $r = -0.29$, optional ratio $r = -0.30$, both relative to mean unsupervised TC.

B Aggregation Sensitivity

The reported tool-calling scores combine the four top-level dimensions via arithmetic mean. Table 16 re-aggregates the same per-turn dimension scores under harmonic mean and minimum-across-dimensions ($n = 385$; six coherence-only records omitted). Corpus-wide means shift substantially (0.911 / 0.851 / 0.781), but per-MCP rankings are highly stable: Spearman $\rho = 0.964$ (Arith. vs Harm.), 0.964 (Arith. vs Min.), 0.929 (Harm. vs Min.). The only swap is Filesystem $\leftrightarrow$ Git near the bottom; the top-3 ordering is identical under all three rules. The schema-complexity correlation (Section 5) holds in direction throughout: Pearson $r = -0.60$ (Arith.), -0.48 (Harm.), -0.50 (Min.) across the seven MCPs, and localizes to the argument sub-dimension ($r = -0.86$ versus $|r| \leq 0.46$ for the other three). Argument correctness is also the dominant failure mode at the input level (60.5% of 448 turns score below 1.0, vs. 29.1% for ordering, 28.6% for selection, 1.3% for usage)—a turn-level fact unaffected by the outer aggregation choice.

C Novelty Summary

Table 17 situates the contributions against prior literature across two gaps: the data generation gap (rows 1–3) and the metric decomposition gap (rows 4–6).

MCP	$\lvert T\rvert$	Arith.	Harm.	Min.
Redis	47	0.966	0.942	0.905
Selenium	56	0.935	0.909	0.814
Elasticsearch	20	0.930	0.890	0.821
Illustrator	64	0.898	0.839	0.745
Slack	16	0.886	0.813	0.737
Filesystem	14	0.876	0.746	0.680
Git	33	0.857	0.763	0.677
Corpus		**0.911**	**0.851**	**0.781**

Table 16: Per-MCP TC means under arithmetic, harmonic, and minimum aggregation across the four dimensions ($n = 385$).

D Pipeline Prompts

This section reproduces the instructional content of the LLM prompts used at each pipeline stage. JSON output templates and tool-spec payloads are abbreviated for space.

D.1 Stage 1: Tool Interpretation

The tool interpreter receives a single MCP tool specification and produces a structured semantic explanation, requested as a JSON object with five named fields.

```
I have a tool that can be called by an agent,
and I could use help understanding what it
does and what it is helpful for.

Tool info:
```json
{tool_info}
```

I need a json in the following format that
can help me thoroughly understand what the
tool is capable of, especially in an
enterprise context. Keep the explanations
grounded within the tool info.

{
  "tool_name": <Tool name here, as given>,
  "what_it_does": <Complete explanation of
      the tool's functionality and what it
      aims to do>,
  "what_it_needs": <What parameters the tool
      needs and how they should be formatted>,
  "why_its_used": <Reasons an agent would
      call this tool; potential use cases>,
  "enterprise_context": <Tags for what aspect
      of an enterprise this could help with>
}
```

D.2 Stage 2: Scenario Generation

Two prompts generate scenarios at different complexity levels from the enriched tool summaries. Both target an "agentic chatbot for enterprise use

Contribution	Prior Literature	This Work
Benchmark construction	Manual curation; or LLM synthesis for training data (Verma et al., 2025; Castellani et al., 2025; Liu et al., 2024)	Automated *evaluation* harnesses from MCP specs, with held-out oracles and mock outputs
Mock output generation	Live API execution (Qin et al., 2024); or neural simulator trained on large corpora (Ren et al., 2025; Castellani et al., 2025)	LLM-prompted generation from spec alone; no live API or trained simulator
Multi-turn grounding	Context-free follow-ups	Mock-data-conditioned turn generation
Tool-calling granularity	Name + parameter match; argument-flow via DAG (Maekawa et al., 2026)	4-dimension decomposition (usage/selection/ordering/arguments)
Argument scoring strategy	Arithmetic mean of parameter scores	Arithmetic mean with judge-enforced cascading penalties on critical errors
Failure taxonomy	Pass/fail	6-category failure type classification

Table 17: Contributions vs. prior literature, organized by thematic gap.

cases" and request scenarios organized by category, each containing an exact `agent_workflow` of function calls with parameters.

Simple scenarios.

```
I'm building an agentic chatbot for enterprise
use cases. Based on the available tool
capabilities below, generate realistic,
straightforward, and commonplace scenarios
organized by category that showcase how
employees would use this chatbot for everyday
tasks. These examples would not require too
many tool calls -- they'll be smaller and
more precise.

Available Tool Capabilities:
{tool_summary}

For each scenario, include the exact function
calls the agent would make using the available
tools.

[JSON format omitted: categories[].scenarios[]
 with title, prompt, agent_workflow[], novelty
 _reason, agent_followup]

Make sure:
 1. Use actual tool names from the available
    capabilities
 2. Function names and parameters are
    structured separately with realistic
    values adhering to the parameter schema
 3. Workflows show logical progression
 4. Scenarios are practical and commonly
    encountered
 5. Agent workflows do the necessary context
    management & tool calls to identify how
    parameters are selected
 6. Provide meaningful agent_followup content
    that makes sense within the context of
    the scenario.
```

Complex scenarios. The complex prompt mirrors the simple prompt but replaces "straightforward, and commonplace" with "novel, and complex", asks for advanced and creative tool usage, and adds two "Make sure" items: *each scenario demonstrates complex, multi-step processes* and *creative combinations of tools that unlock new capabilities*.

Coverage hint and follow-up. A coverage suffix appended to the initial prompt instructs: *"IMPORTANT: Ensure broad coverage across ALL available tools. Every tool listed above should appear in at least one scenario's agent_workflow. There are {N} tools total — design scenarios that collectively exercise all of them."* If tools remain uncovered after the first round, a follow-up prompt requests additional scenarios for the named uncovered tools, allowing combination with previously covered tools in multi-step workflows.

D.3 Stage 3: Mock Output Generation

```
You are a Mock Tool Output Generator for
synthetic agent workflow data. Your task is
to generate realistic mock tool outputs that
complete synthetic scenarios.

You will be given an initial prompt (and
maybe a description of what the aim of the
prompt is & why the scenario is of interest).
Then, you will be given an agent workflow.
This will detail:
  (1) A function call w/ parameters and
  (2) A quick explanation of what the tool
      does.

Finally, you will have some example function
calls with their respective outputs OR a JSON
schema object describing the output
structure. Use this to guide formatting for
the final mock tool output.

[JSON output: mock_workflow[] with function
 _name, parameters, quick_explanation,
 mock_output, confidence; plus expected
 _response that references specific mock data]

### CONFIDENCE LEVEL GUIDELINES
"high":   concrete example for THIS specific
```

```
          function was provided
"medium": no example for this function, but
          similar functions have examples
"low":    no example output provided for
          this function

### CRITICAL INSTRUCTIONS
 1. Concrete Data: replace placeholders (e.g.
    "{user_id}" or "XYZ") with realistic,
    specific values.
 2. Realism & Diversity: reflect how a real
    system would respond; incorporate diverse
    names, global locations, and varied data
    points.
 3. Formatting: strictly adhere to provided
    reference examples or JSON schema.
 4. Expected response references concrete
    mock data (names, IDs, counts, statuses,
    dates) and reflects the full workflow,
    not just the last call.
```

D.4 Stage 4: Multi-Turn Expansion

The multi-turn expansion prompt is constructed in parts: a preamble, scenario and tool-result context, key principles, task instructions, worked examples of good vs. bad turn splitting, quality guidelines, and a JSON output template. The principle set differs between *mock-data-grounded* mode (when synthetic outputs exist) and *standard* mode.

```
You are a Natural Conversation Flow Analyzer
for enterprise agent interactions. Your task
is to take a scenario and intelligently break
it into natural conversation turns that
reflect how real employees would interact
with an agent.

# Key Principles (mock-data-grounded mode)
 1. Use Available Tool Results to inform
    realistic follow-up questions and workflow
    decisions.
 2. Data-Driven Breakpoints from the actual
    data returned by tools.
 3. Realistic User Reactions to the specific
    data shown.
 4. Progressive Data Exploration: each turn
    builds on prior tool outputs.
 5. Split at Phase Boundaries when distinct
    phases exist (e.g., information-gathering
    then acting on it).
 6. Group Related Operations Within a Phase:
    batch repeated operations (e.g., 3
    lookups) into a SINGLE turn.
 7. Self-Contained Turns: agent_followup
    reports concrete results, NOT narration of
    future actions.
 8. Complete the Full Workflow: every tool
    call from the initial workflow appears in
    exactly one turn.
 9. CRITICAL CONSTRAINT - Use Only Existing
    Functions: same function names and
    parameter structures as the initial
    workflow.

[Worked examples follow: BAD over-granular
 splitting, BAD no-splitting, GOOD batched-
 within-phases. JSON format omitted: turns[]
 with title, prompt, agent_workflow[]
 including mock_output and confidence,
 novelty_reason, agent_followup]
```

In *standard* mode (no mock outputs available), the principles drop the data-driven breakpoint and tool-result references, and the JSON template omits the `mock_output` and `confidence` fields. The phase-boundary, batching, self-contained-turn, and function-reuse constraints are preserved across both modes.

E Evaluation Prompts

The quality assessment framework (Section 4) uses two LLM-as-judge prompts: one for tool-calling correctness and one for conversational coherence. Both operate in unsupervised mode (no reference answer). Tables 18 and 19 detail the aspects measured, their definitions, and scoring scales.

E.1 Tool-Calling Evaluation

The tool-calling prompt evaluates across four dimensions, each with scored sub-dimensions on a 0–10 scale. Sub-scores are normalized to 0–1; the selection, ordering, and argument dimension scores are the arithmetic mean of their sub-scores, while the usage dimension score is taken directly from its necessity sub-score. The four top-level dimension scores are then combined via arithmetic mean. Cascading penalties on argument sub-scores are enforced through prompt instructions to the judge (Table 18, footnote).

E.2 Coherence Evaluation

The coherence prompt evaluates across five dimensions on a 1–3 scale, normalized to 0–1 and aggregated via arithmetic mean. Each dimension checks for specific failure manifestations.

F Structured Output Schemas

Each pipeline stage uses Pydantic models as structured output schemas, constraining the LLM to produce well-formed JSON at every step.

G Score Distributions and Per-Domain Analysis

G.1 Overall Score Distributions

Figure 3 shows the distribution of tool-calling and coherence scores across all 391 evaluation records. The tool-calling distribution is concentrated in the upper range (31.7% perfect scores, 2.3% below

Dimension	Sub-dimension	Definition	Scale
Usage	Necessity	Was a tool actually needed, or could the assistant answer directly?	0–10
	Overuse detection	Are there redundant or unnecessary tool calls?	0–10
Selection	Correctness	Do the selected tools match the task described by the user?	0–10
	Specificity	Was the most specific tool chosen when alternatives exist?	0–10
	Completeness	Are all tools needed to fully address the query called?	0–10
Ordering	Sequence logic	Is the execution order logical; do later calls build on earlier ones?	0–10
	Dependency handling	Are inter-tool dependencies respected (output $\rightarrow$ input)?	0–10
	Execution efficiency	Could reordering improve efficiency?	0–10
Arguments	Completeness	Are all required parameters provided?	0–10
	Name accuracy	Do parameter names match schemas exactly (case-sensitive)?	0–10
	Value accuracy	Are values correct and grounded in the user query or prior tool outputs?	0–10
	Type compliance	Do parameter values match expected data types?	0–10
	Format compliance	Do values follow expected formats (dates, enums, patterns)?	0–10
	Relevancy	Are there any extra or invalid parameters not in the schema?	0–10

Table 18: Tool-calling evaluation dimensions and sub-dimensions. Ordering is marked not applicable for single tool calls and excluded from aggregation. **Cascading rules** (enforced via prompt instructions to the judge): when the judge identifies a wrong parameter name (score ≤ 2), it is instructed to assign near-zero scores to the dependent argument sub-dimensions (value, type, format); a missing required parameter triggers the same cascade; a wrong value (score ≤ 3) cascades to type, format, and relevancy. Values from prior tool outputs in chained calls are not penalized.

0.5). The coherence distribution is left-skewed with a mode near 0.9.

G.2 Per-Domain Analysis

Figures 4–10 show the scenario category breakdown (simple vs. complex), workflow length distribution, tool co-occurrence graph, and sequential adjacency graph for each MCP specification. Co-occurrence graphs show which tools appear in the same scenario (undirected); adjacency graphs show tool $A \rightarrow$ tool B transition patterns within ordered workflows (directed).

Dimension	Definition	Failure manifestations
Context retention	How well the response maintains and uses information from conversation history	`asks_again`, `forgets_preferences`, `pronoun_confusion`, `contradicts_self`, `loses_thread`, `ignores_corrections`
Logical flow	How well the response follows logically from previous turns and maintains coherent progression	`topic_shift`, `non_sequitur`, `poor_transitions`, `breaks_causality`, `temporal_confusion`
Completeness	How thoroughly the response addresses all parts of the user's query	`cuts_off`, `partial_answer`, `too_shallow`, `ignores_constraints`, `missing_key_info`, `no_actionable_advice`
Conciseness	How efficiently the response communicates without unnecessary repetition	`repeats_directly`, `rephrases_same_point`, `too_much_fluff`, `over_explains`, `excessive_caution`
Topic relevance	How well the response stays focused on the user's query and conversation topic	`complete_topic_shift`, `misses_main_point`, `too_generic`, `hallucination`, `wrong_question`

Table 19: Coherence evaluation dimensions. Each dimension is scored on a 1–3 scale: **Good** (3) = no manifestations detected; **Adequate** (2) = 1–2 minor manifestations; **Poor** (1) = 3+ manifestations or critical failures. Context retention is optional (excluded when no conversation history exists).

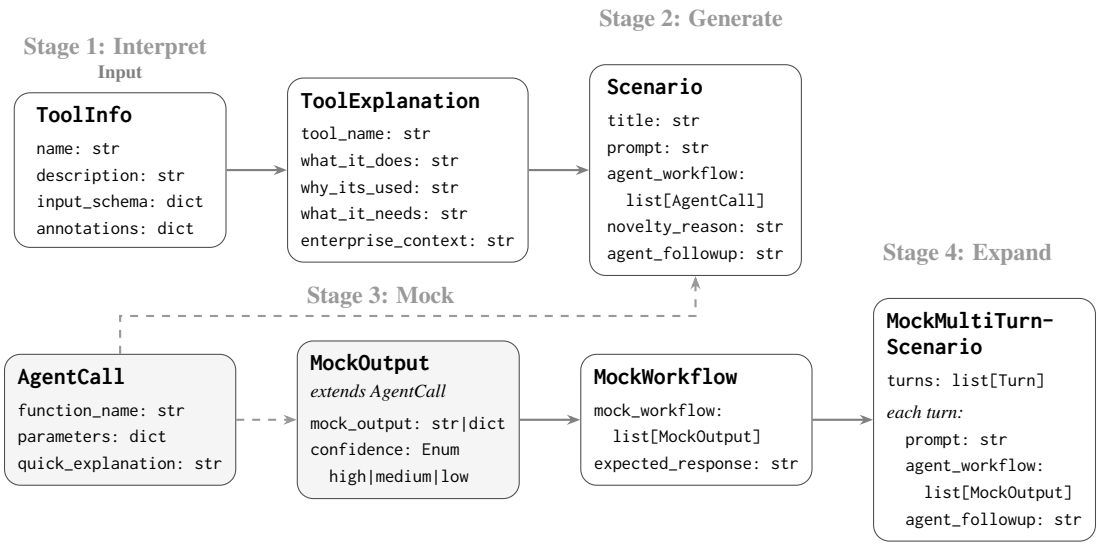
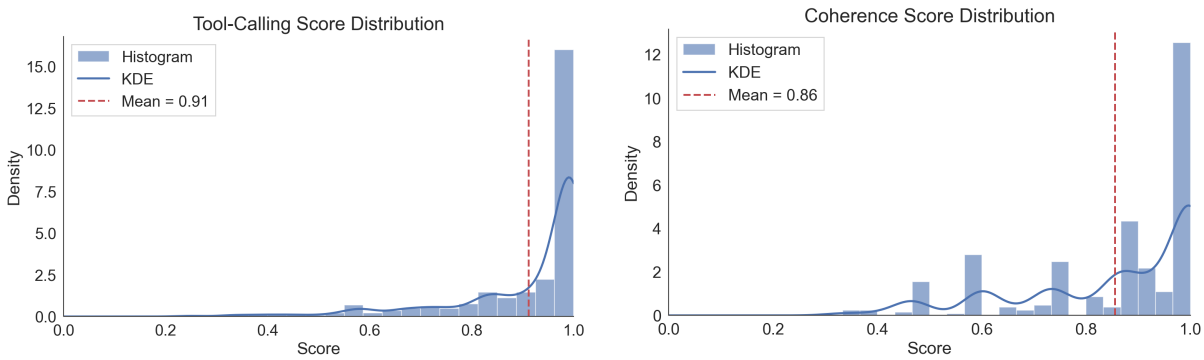


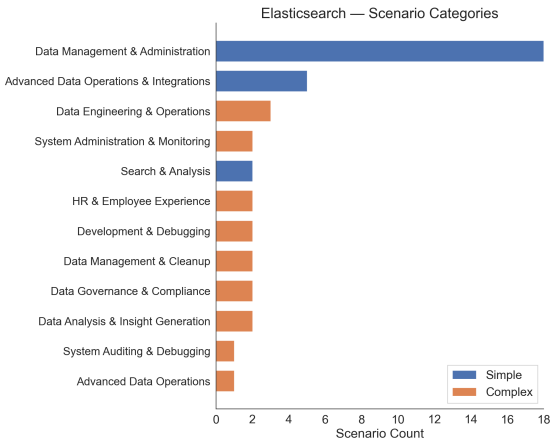
Figure 2: Structured output schemas across the four pipeline stages. Solid arrows indicate data flow; dashed arrows indicate schema inheritance (`MockOutput` extends `AgentCall`, which is referenced by `Scenario`).



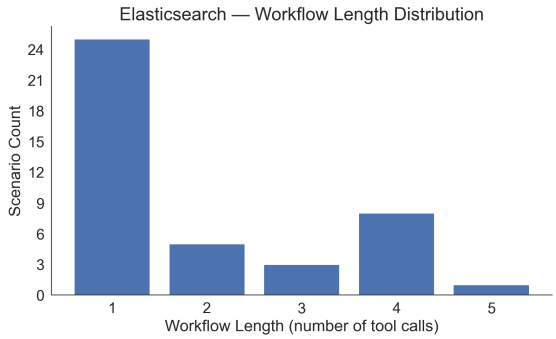
(a) Tool-calling score distribution with KDE.

(b) Coherence score distribution with KDE.

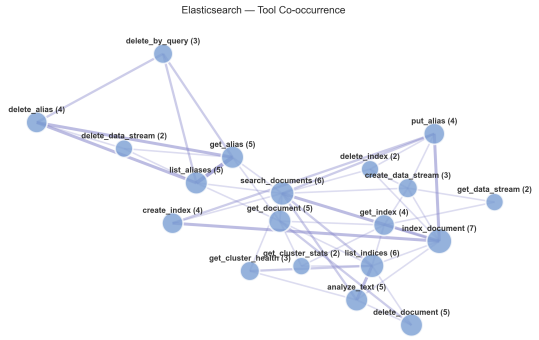
Figure 3: Score distributions across all 391 evaluation records.



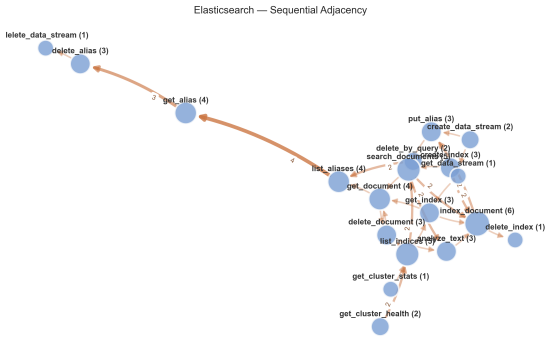
(a) Scenario categories.



(b) Workflow length distribution.

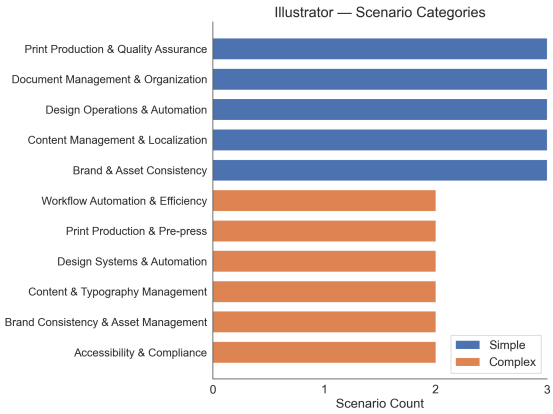


(c) Tool co-occurrence graph.

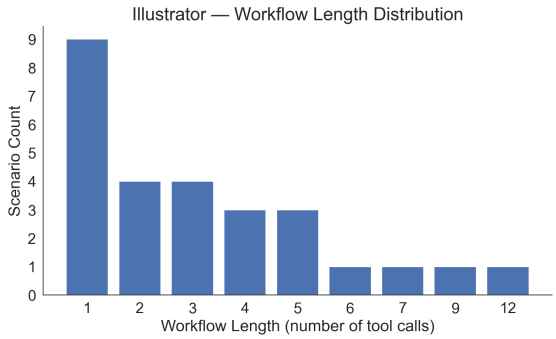


(d) Sequential adjacency graph.

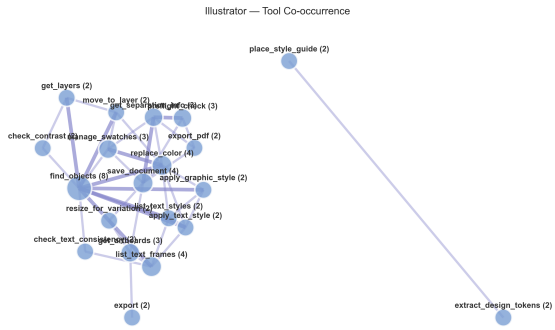
Figure 4: Elasticsearch (20 tools): scenario categories, workflow lengths (mean 1.9 calls), tool co-occurrence, and sequential adjacency.



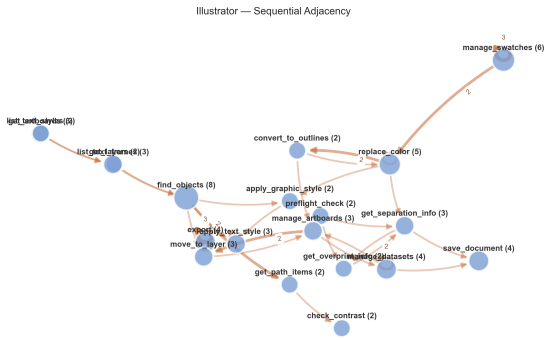
(a) Scenario categories.



(b) Workflow length distribution.

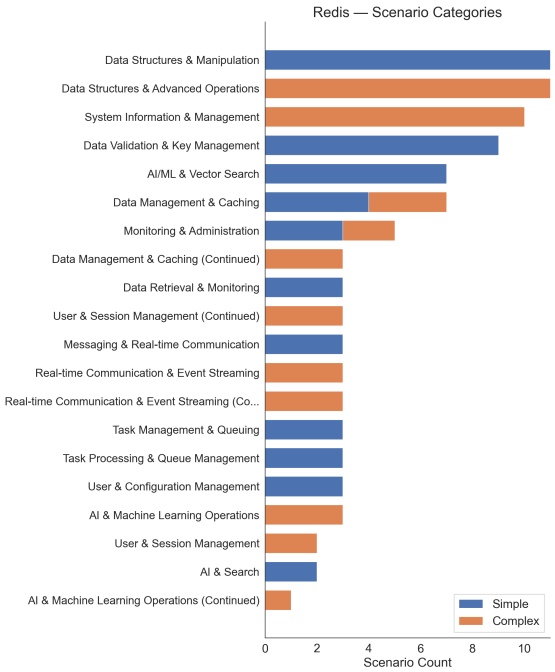


(c) Tool co-occurrence graph.

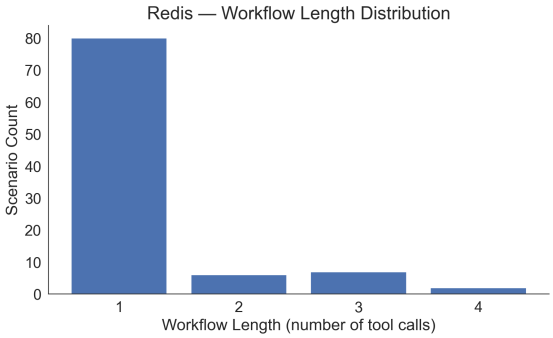


(d) Sequential adjacency graph.

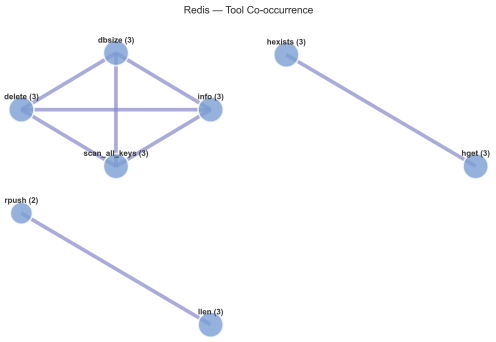
Figure 5: Illustrator (64 tools): scenario categories, workflow lengths (mean 3.3 calls, max 12), tool co-occurrence, and sequential adjacency.



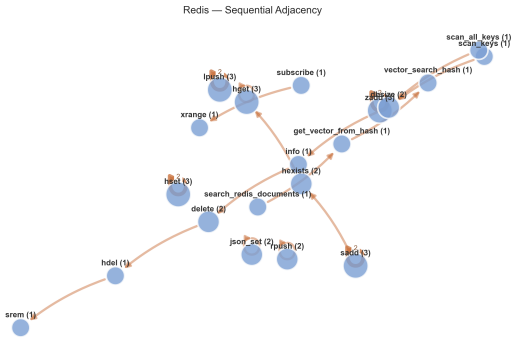
(a) Scenario categories.



(b) Workflow length distribution.

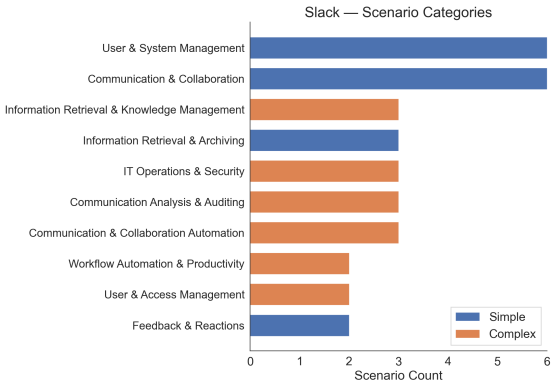


(c) Tool co-occurrence graph.

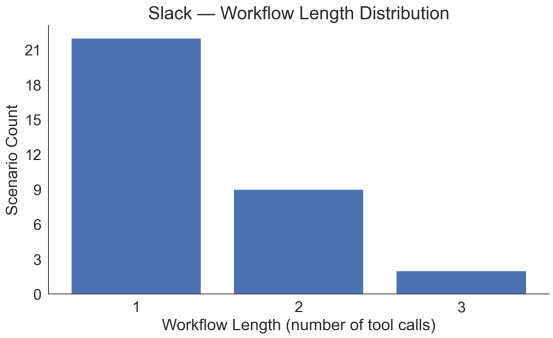


(d) Sequential adjacency graph.

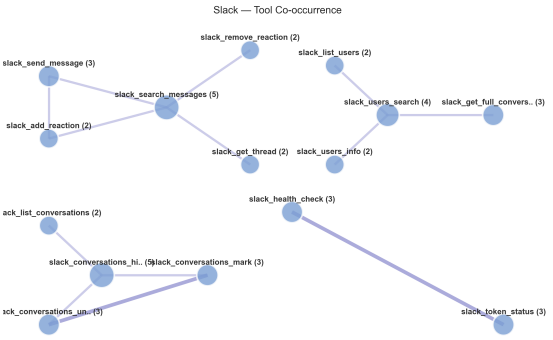
Figure 6: Redis (47 tools): scenario categories, workflow lengths (mean 1.3 calls), tool co-occurrence, and sequential adjacency.



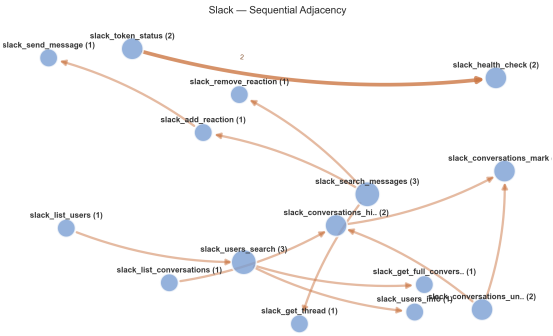
(a) Scenario categories.



(b) Workflow length distribution.

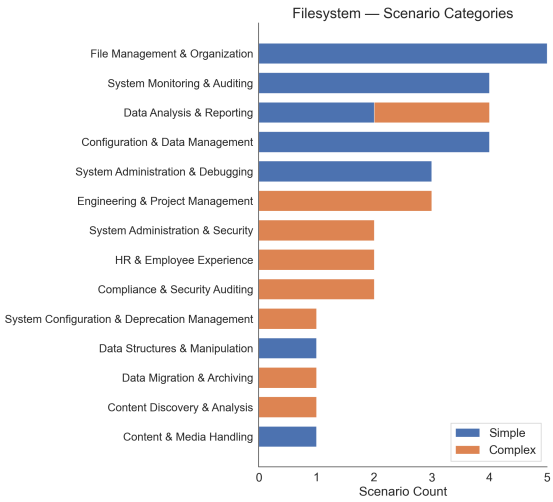


(c) Tool co-occurrence graph.

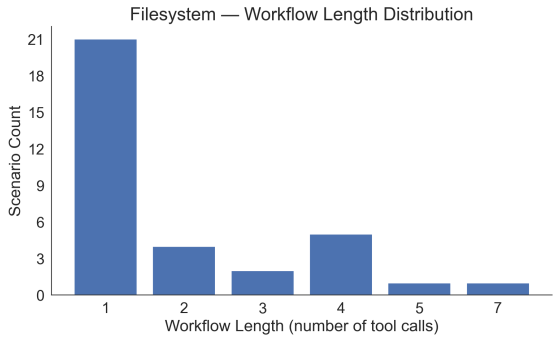


(d) Sequential adjacency graph.

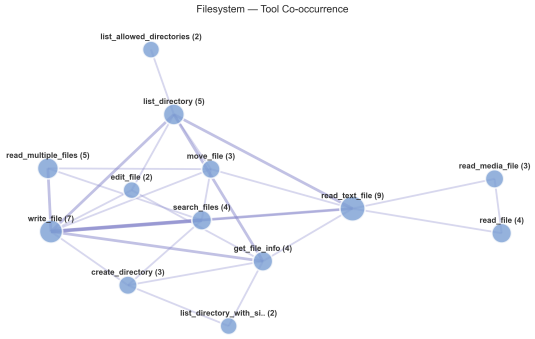
Figure 7: Slack (16 tools): scenario categories, workflow lengths (mean 1.4 calls), tool co-occurrence, and sequential adjacency.



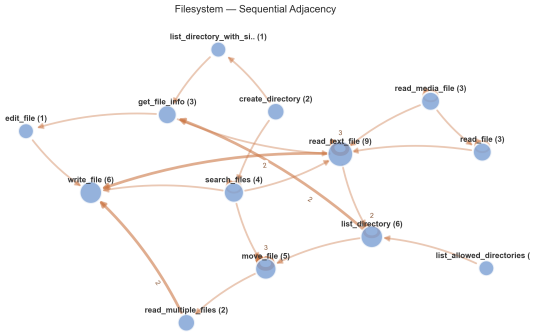
(a) Scenario categories.



(b) Workflow length distribution.



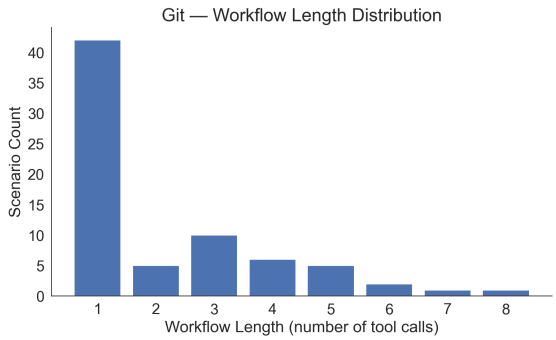
(c) Tool co-occurrence graph.



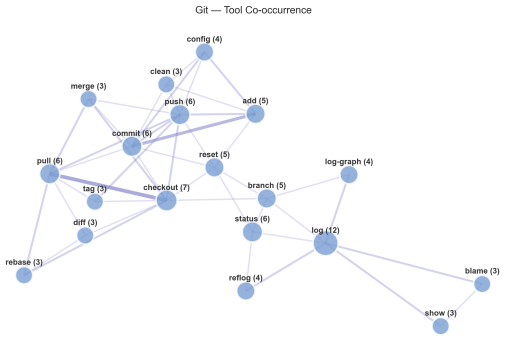
(d) Sequential adjacency graph.

Figure 8: Filesystem (14 tools): scenario categories, workflow lengths (mean 2.0 calls), tool co-occurrence, and sequential adjacency.

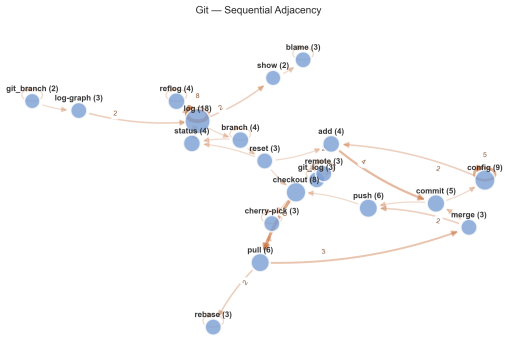
(a) Scenario categories.



(b) Workflow length distribution.

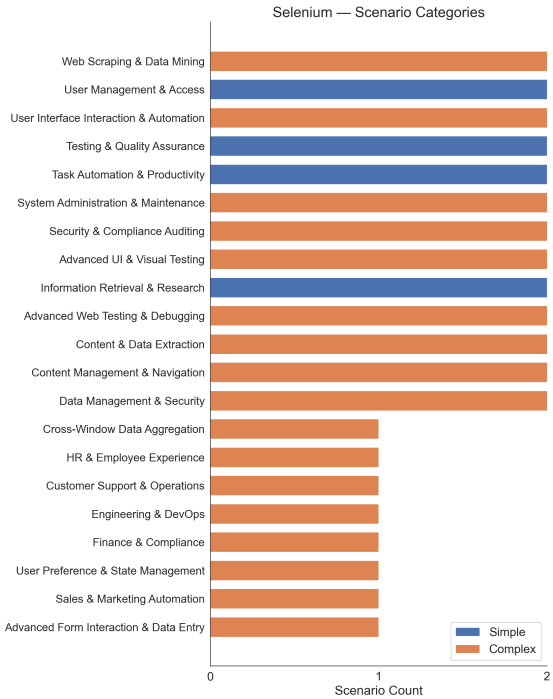


(c) Tool co-occurrence graph.

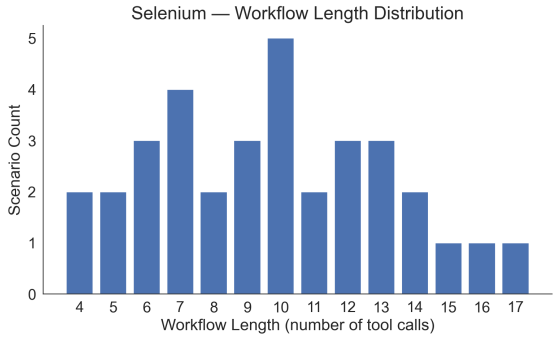


(d) Sequential adjacency graph.

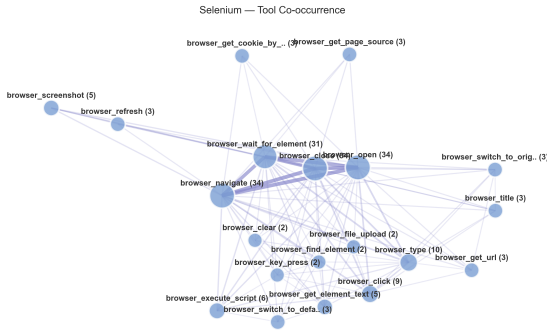
Figure 9: Git (33 tools): scenario categories, workflow lengths (mean 2.2 calls), tool co-occurrence, and sequential adjacency.



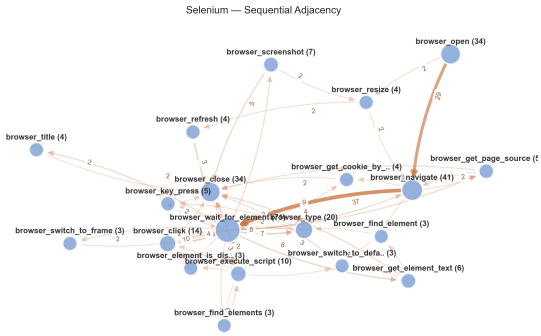
(a) Scenario categories.



(b) Workflow length distribution.



(c) Tool co-occurrence graph.



(d) Sequential adjacency graph.

Figure 10: Selenium (56 tools): scenario categories, workflow lengths (mean 9.7 calls, max 17), tool co-occurrence, and sequential adjacency.