Coverage-Aware Test Generation for Conversational AI Agents

Mridul Katta [1] **Darin Glatt** [1]

Abstract

Testing multi-capability conversational AI agents requires diverse, high-coverage test scenarios spanning the agent's full intent space. Existing synthetic data tools generate scenarios from context documents in a single pass, causing *intent collapse*—the generated set concentrates on the most prominent topic, leaving the majority of agent capabilities untested. We present **Scenario Studio (SS)**, a config-driven pipeline that prevents intent collapse through *coverage-aware orchestration*: a PolicyGraph models the agent's full capability space, a CoverageTracker monitors real-time intent coverage, and gap-directed generation steers each round toward uncovered areas. SS also serves as a *pluggable orchestration layer*—any open-source generation or red-teaming tool can be integrated as a backend module, immediately benefiting from SS's compiled context and coverage steering.

We benchmark SS against DeepEval Synthesizer and two GPT-prompting baselines across two production agents: **Agent-A**, a diverse general-support assistant spanning 7+ intent families (127 ground-truth queries), and **Agent-B**, a narrow ride-booking B2B agent (106 queries). SS achieves 86.6% ground-truth coverage on the diverse agent versus 15.7% for DeepEval—a **5.5× improvement** with non-overlapping 95% bootstrap confidence intervals ($p < 0.05$). With a comparable frontier model, SS outperforms premium GPT-5.2 baselines by +14.9pp, demonstrating that *architecture matters more than model capability*. A dual-judge study confirms 89.0–95.3% per-item agreement with identical tool rankings, and all gains on the diverse agent are statistically significant under both judges. SS also generates adversarial scenarios across 12+ attack categories via hybrid red-teaming backends (native + DeepTeam + PromptFoo) and supports consolidation that removes 31–37% of scenarios while slightly improving coverage.

[1]Uber Technologies, Inc., USA. Correspondence to: Mridul Katta <mridul.katta@uber.com>.

Published at the Second Workshop on Agents in the Wild: Safety, Security, and Beyond (AIWILD) at ICML 2026. Copyright 2026 by the author(s).

1. Introduction

LLM-based conversational agents are increasingly deployed as general-purpose assistants handling diverse capabilities within a single interface. Evaluating these agents requires test scenarios covering the full breadth of capabilities—not just the dominant use case.

A natural approach is to use LLMs to *generate* synthetic test scenarios from agent documentation. Tools such as DeepEval Synthesizer (1) and simple GPT prompting take context documents as input and produce test cases in a single pass. However, we observe that this approach suffers from a fundamental limitation we term **intent collapse**: the LLM gravitates toward the richest content cluster in the input, producing scenarios overwhelmingly concentrated on a single intent family.

We frame intent collapse as a **distribution collapse** problem. When an LLM generates test scenarios from documents, it samples from an implicit distribution over the intent space. Without explicit coverage guidance, this distribution collapses to a low-entropy mode centered on the most prominent documentation cluster—analogous to mode collapse in generative models. In our experiments, DeepEval generates 98%+ ride-booking scenarios for an agent supporting 7+ intent families.

We present **Scenario Studio (SS)**, a config-driven pipeline that prevents intent collapse through *coverage-aware orchestration*—a structured exploration mechanism that:

1. **Models** the full intent space via a PolicyGraph extracted from agent code;
2. **Tracks** real-time coverage as scenarios are generated;
3. **Steers** subsequent generation toward uncovered intents.

This is analogous to coverage-guided fuzzing in software engineering (17), where instrumentation tracks which code paths have been exercised and mutation targets uncovered paths. SS applies this principle to the intent space of conversational agents.

SS is also a **pluggable orchestration layer**: third-party tools (DeepEval, PromptFoo, DeepTeam) integrate as backend modules, gaining SS's compiled context and coverage steering. We show that even providing SS's context alone to DeepEval improves its coverage from 15.7% to 50.8% (+35pp).

Our contributions:

- A coverage-aware generation architecture achieving 86.6% coverage versus 15.7% for DeepEval—a 5.5× improvement;
- Evidence that intent collapse is structural, not a sample-size issue, framed as distribution collapse with a coupon-collector-style argument;
- A controlled experiment cleanly separating context quality (+35pp) from coverage-aware orchestration (+36pp);
- A cross-model study showing architecture beats model: SS with a frontier-tier model outperforms GPT-5.2 baselines by +14.9pp;
- A pluggable architecture enabling any open-source tool to benefit from SS's context compilation and coverage steering;
- A dual-judge validation study confirming 89.0–95.3% agreement, with bootstrap CIs establishing statistical significance of all Agent-A gains ($p < 0.05$).

2. Related Work

Synthetic Test Data Generation. DeepEval Synthesizer (1) generates test cases from context documents using single-pass generation with optional evolution operators. DSPy (2) optimizes prompts rather than generating test scenarios. PromptFoo (3) provides red-teaming and evaluation but lacks coverage-aware generation. Self-Instruct (4) and Bonito (5) generate instruction-following data through iterative refinement, but do not model target coverage. Recent work on diverse synthetic prompt generation—CoT-Self-Instruct (6), which seeds generation with chain-of-thought reasoning over existing tasks, and BARE (7), which combines base and instruction-tuned models to widen the diversity of generated data—shares our motivation of avoiding mode collapse, but does not address the agent-specific intent space that is the focus of this work. None of these tools model the agent's intent space or track coverage during generation.

LLM-as-Judge Evaluation. G-Eval (8) and MT-Bench (9) establish the paradigm of using LLMs as evaluators. We adopt a retrieve-then-judge approach to reduce LLM calls while maintaining accuracy.

Agent Testing and Evaluation. AgentBench (10) and GAIA (11) provide static benchmarks for specific agent capabilities. HELM (12) offers holistic evaluation across multiple dimensions. AgentBoard (13) provides multi-turn evaluation with progress tracking. ToolBench (14) and TaskBench (15) benchmark tool-use capabilities. IntellAgent (16) uses policy-driven evaluation via behavior graphs, conceptually similar to our PolicyGraph but focused on *evaluation* (simulating and scoring dialogues) rather than *generating* a coverage-calibrated scenario pack from an agent's source context. We therefore treat it as related framing rather than a drop-in generation baseline: our baselines (DeepEval Synthesizer and GPT prompting) are context-to-scenario generators directly comparable to SS, whereas slotting IntellAgent into the same harness would require re-implementing it as a generator backend—an integration we view as complementary future work given SS's pluggable design. Our work bridges the gap between generation and evaluation through coverage-aware orchestration.

Coverage-Guided Testing. Coverage-guided fuzzing (AFL (17)) uses code instrumentation to steer mutation toward uncovered paths. Property-based testing (QuickCheck (18)) generates structured inputs guided by type specifications. We adapt this principle to LLM-based scenario generation, using the PolicyGraph as the "coverage map" and the CoverageTracker as the instrumentation layer.

3. System Architecture

SS is a config-driven pipeline that takes a `PipelineSpec`—an agent-agnostic configuration that wires together generators, evolution operators, evaluation hooks, and other pipeline settings—and produces diverse test scenarios with simulation-ready prompts. The `PipelineSpec` contains an agent-specific `DomainSpec` block that declares the target agent's intent space, policies, and source references; the rest of the PipelineSpec is reusable across agents. Figure 1 shows the end-to-end architecture.

3.1. Coverage-Aware Generation Loop

Algorithm 1 formalizes the core orchestration. The PolicyGraph $\mathcal{G}$ models the agent's full intent space as a directed graph where nodes represent testable policies and edges represent dependencies. The CoverageTracker $\mathcal{C}$ maintains a real-time ledger of covered intents.

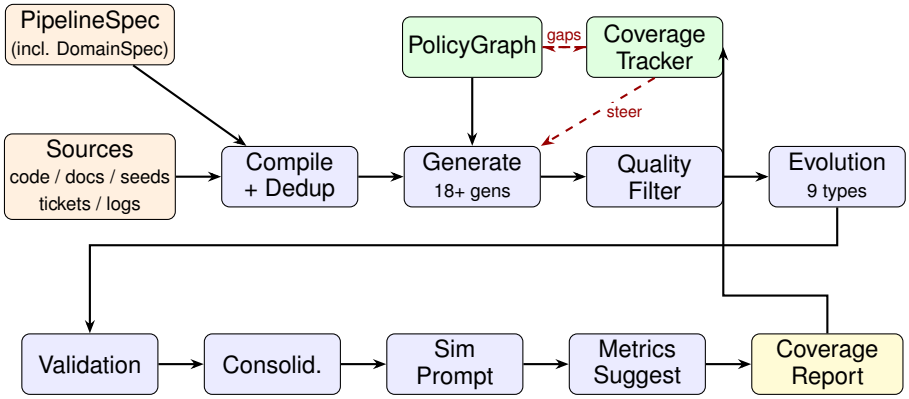

Figure 1. SS pipeline architecture. The agent-agnostic **PipelineSpec** (containing the agent-specific **DomainSpec** block) plus heterogeneous sources (code, docs, seeds, tickets, production logs/traces) feed a *Compile* stage. The **PolicyGraph** models the agent's full intent space; the **CoverageTracker** monitors covered intents and steers generation toward gaps (red dashed arrows), preventing intent collapse. Later stages validate, consolidate, emit simulation prompts, design an agent-specific evaluation framework (*Metrics Suggest*), and produce a final *Coverage Report* that highlights uncovered behaviours so developers can enrich the source context and re-run.

Algorithm 1 Coverage-Aware Generation

Input: DomainSpec $\mathcal{D}$, Sources $\mathcal{S}$, max rounds R, generators $G_1, \ldots, G_m$
Output: ScenarioPack $\mathcal{P}$

$\mathcal{G} \leftarrow$ BuildPolicyGraph($\mathcal{D}$) $\mathcal{P} \leftarrow$ Compile($\mathcal{S}, \mathcal{D}$)
$\mathcal{C} \leftarrow$ InitTracker($\mathcal{G}$) $\mathcal{C}$.Update($\mathcal{P}$) **for** $r \leftarrow 1$ **to** R **do**
  $\mathcal{U} \leftarrow \mathcal{C}$.GetUncovered() **if** $|\mathcal{U}| = 0$ **then**
    break
  $w \leftarrow$ AllocateSlots($\mathcal{U}, G_1, \ldots, G_m$) **for** $i \leftarrow 1$ **to** m **do**
    $\mathcal{P}_i \leftarrow G_i$.Generate($w_i, \mathcal{U}, \mathcal{D}$) $\mathcal{P}_i \leftarrow$ QualityFilter($\mathcal{P}_i$) $\mathcal{P} \leftarrow \mathcal{P} \cup$ Dedup($\mathcal{P}_i, \mathcal{P}$)
  $\mathcal{C}$.Update($\mathcal{P}$)
$\mathcal{P} \leftarrow$ Evolve($\mathcal{P}$) $\mathcal{P} \leftarrow$ Consolidate($\mathcal{P}, \mathcal{C}$) $\mathcal{P} \leftarrow$ GenerateSimPrompts($\mathcal{P}, \mathcal{D}$) **return** $\mathcal{P}$

Why single-pass generation fails. As intuition (not a formal bound), consider an agent with k distinct intent families and documentation where one family occupies fraction $p_{\max} \gg 1/k$ of the content. If single-pass generation samples intents roughly proportional to documentation density, covering all intents takes on the order of a coupon-collector-style number of samples that grows with $1/p_{\min}$, whereas coverage-aware generation with explicit gap targeting needs far fewer directed samples since each round targets the largest remaining gap. We present this as a conceptual framing rather than a rigorous analysis, but our experiments are consistent with it: DeepEval at 250 scenarios achieves only 27.6% coverage (gpt-4o-mini), while SS at 228 scenarios reaches 86.6% (gemini-2.5-flash).

3.2. DomainSpec, PolicyGraph, and CoverageTracker

DomainSpec. The agent-specific `DomainSpec` is a declarative YAML block that captures everything SS needs to know about *this* agent, separate from the reusable pipeline configuration. It records: (i) the agent type, a short natural-language description, and supported languages; (ii) the intent families and testable policies that define the capability surface; (iii) the typed *slots* the agent must elicit, each with an extraction policy—whether the slot is required, what to do when it is missing (e.g., `ask` vs. `proceed`) or vague (e.g., `clarify`), example "vague" phrasings to recognize, and concrete value variations; (iv) hard *constraints* written as must-not invariants (safety, policy, and feasibility rules the agent may never violate); (v) *edge cases* as `(tag, description, expected_behavior)` triples; and (vi) optional steering hints (priority intents, exclusions). These turn an informal capability description into a machine-checkable target: the slots, constraints, and edge cases seed both the PolicyGraph nodes and the downstream validation rules, while the source references drive context compilation. A DomainSpec can be authored by hand or bootstrapped with the `--discover-domain` flag, which drafts an initial version from the available sources for the user to review and refine.

PolicyGraph. The PolicyGraph is an LLM-extracted behavior graph where nodes represent testable policies (e.g., "ride_booking," "restaurant_search") and edges represent co-occurrence or dependency relationships. It serves two purposes: (1) providing generation prompts with the full intent taxonomy; (2) enabling intent-level coverage measurement.

PolicyGraph extraction quality. Because downstream coverage is measured against PolicyGraph nodes, graph quality directly affects the reported numbers, so SS treats

the graph as a reviewable artifact rather than a hidden intermediate. After extraction, the node/edge set is emitted in the coverage report for human review, and missing or spurious nodes are corrected by editing the `DomainSpec` and re-running. For both agents in this study the authors manually audited the extracted nodes against the known intent families and found the families fully represented; node-level errors we observed were predominantly over-splitting (one intent surfaced as two near-synonymous nodes), which the synonym-aware grouping in consolidation collapses, rather than missing intents. Two sensitivity considerations follow: a *missing* node cannot be steered toward and would deflate coverage (a conservative bias), while an *over-split* node inflates the denominator but is corrected at consolidation; in both cases the gap-directed loop and the human-reviewed coverage report surface the discrepancy. A systematic study of extraction accuracy across many agents is left to future work.

CoverageTracker and slot allocation. The CoverageTracker maintains a per-node coverage count. After each generation batch, it identifies uncovered (or under-covered) nodes and feeds them to the allocation function, which distributes a fixed per-round generation budget across the available generators. Allocation is proportional to each node's remaining gap and weighted by an *affinity score*—a heuristic match between a generator's strengths (e.g., multi-turn, adversarial, constrained) and a node's policy type—so that uncovered intents are routed to the generators most likely to cover them, with a floor that guarantees every uncovered node receives at least one slot per round.

3.3. Pluggable Architecture

SS provides two integration surfaces for third-party tools:

SourceContextDump. SS compiles a structured context package—domain spec, intent taxonomy, agent code analysis, documentation—that any external tool can consume directly. When DeepEval receives this context instead of raw documents, its coverage jumps from 15.7% to 50.8% (+35pp), confirming that SS's context compilation alone is a significant contribution.

Backend plugins. External tools plug into SS's pipeline as generation or red-teaming modules:

- **DeepEval Synthesizer**: alternative generation backend
- **DeepTeam** (19): red-teaming backend alongside SS-native adversarial generators
- **PromptFoo**: evaluation and red-teaming backend

All backends benefit from coverage-aware steering and the compiled context, making SS a *force multiplier* rather than a replacement.

3.4. Evolution and Consolidation

The evolution stage applies 9 mutation types (reasoning, constrained, hypothetical, deepening, comparative, multi-context, complicate_input, concretizing, in_breadth) to create harder variants. Evolution doubles average scenario depth and edge-case count without sacrificing intent breadth.

Consolidation uses LLM-based intent extraction and greedy set-cover to remove redundant scenarios—reducing pack size by 31.0%–37.3% while slightly improving coverage (+0.8–+0.9 pp).

3.5. Production Style Profiler

A common failure mode of LLM-generated test data is *stylistic drift*: scenarios are grammatically polished and verbose, whereas real users send short, casual, abbreviation-heavy queries. SS addresses this with a cross-cutting *Production Style Profiler* that builds a `ProductionStyleProfile` from (in priority order) (i) production traces, (ii) curated seed CSVs, (iii) DomainSpec + documents with an LLM inferring plausible phrasing, or (iv) manual hints. The profile captures tone (casual/formal/mixed), average and 90th-percentile input length, a bank of representative phrasings, common abbreviations, and language mix, and is injected into every prompt that emits user-facing text (extraction, generation, evolution). This keeps synthetic scenarios grounded in the real user distribution rather than in LLM-typical prose, and degrades gracefully on green-field agents with no traffic yet.

3.6. Metric Suggestion

Covering the intent space is only half of the evaluation problem: once a pack is generated, teams still have to decide *what to measure*. SS automates this via three LLM calls over the full pack context: **(i) Agent analysis**—classifies the agent into one of five archetypes (*transactional, informational, decisional, generative, hybrid*) and proposes 3–7 weighted evaluation dimensions with failure modes and headline-score gates (e.g., "Agent Correctness ≥ 0.90 PASS"); **(ii) Judge design**—emits self-contained LLM-judge prompts with typed output fields, rubrics, and calibration examples drawn from actual scenarios in the pack; **(iii) Gap analysis**—audits coverage across policy-graph, taxonomy, regression, safety, and edge-case axes. Optional `EvaluationGuidelines` from the agent owner (priorities, weight hints, exclusions) steer the analysis as high-signal input. A single SS run thus yields both a coverage-calibrated scenario pack and the evaluation framework for measuring agent behaviour on it.

4. Experimental Setup

4.1. Agents Under Test

We evaluate on two production conversational agents. **Agent-A** is a multi-capability general-support assistant serving end-user requests across 7+ intent families: ride booking, restaurant search, flight status, nearby places, weather, package delivery, and account management (127 ground-truth queries). Its breadth is representative of production consumer-facing assistants where intent collapse is most damaging. **Agent-B** is a narrow business-to-business (B2B) ride-booking agent that handles 2–3 ride-related intents—booking, modification, and status (106 ground-truth queries). The limited intent space provides a low-ceiling control that isolates the effect of coverage-aware orchestration from raw problem difficulty.

Ground-truth query sets. For each agent we use a held-out set of ground-truth (GT) queries (127 for Agent-A, 106 for Agent-B) as the coverage target. These sets were authored by engineers and product managers from product usage and capability specifications, and—critically—they predate Scenario Studio: they were curated as the agents' own regression/evaluation suites *before* SS existed and were never used as input to any tool under comparison. This temporal separation rules out leakage from the GT set into SS's generation context, so a tool can only "cover" a GT query by independently generating a scenario with the same intent. The GT queries span the full intent set of each agent (e.g., for Agent-A, all 7+ families including long-tail intents such as package delivery and weather) and include short, casual, and multilingual phrasings drawn from production traffic. The sets are derived from a proprietary production system and are not publicly released; we report their construction here so the coverage metric can be interpreted, and we evaluate against public conversational-agent benchmarks as future work (§7).

4.2. Tools Compared

- **SS**: Our coverage-aware pipeline.
- **DeepEval Synthesizer** (1): styled, styled+evolved, and +SS Context variants.
- **GPT Simple**: Single-shot prompting for diverse test scenarios.
- **GPT Structured**: Structured-output prompting with intent schema.

4.3. Evaluation Metrics

D1 Ground-Truth Coverage: For each ground-truth query, an LLM judge determines whether any generated scenario covers the same intent, using a retrieve-then-judge approach.

D2 Cross-Tool Coverage: Pairwise measurement of what percentage of one tool's scenarios are covered by another.

Dual-Judge Validation: D1 run with both gemini-2.5-flash and gpt-5.2 to validate robustness.

4.4. Cross-Model Experiments

We run two full experiments: (1) **All-Gemini** (gemini-2.5-flash for all tools); (2) **All-GPT-4o-mini** (cost-sensitive model), with GPT baselines additionally using frontier models for comparison.

5. Results

5.1. D1 Ground-Truth Coverage

Table 1 and Figure 2 present the main results. SS achieves the highest coverage across both agents, both models, and both judges. On the diverse Agent-A, SS reaches 86.6% versus 71.7% for the best baseline (+14.9pp). With the cost-sensitive model, the gap widens: SS at 78.7% versus 27.6% for DeepEval (+51.1pp). All Agent-A differences are **statistically significant**: SS's 95% CI lower bound (71.7) exceeds every baseline's upper bound ($\leq$38.6) under both judges (see §6.3).

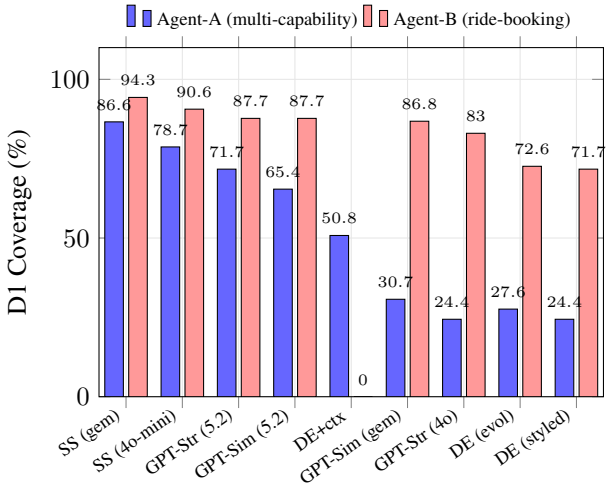

Figure 2. D1 ground-truth coverage across all tools and models (gemini judge). SS with gpt-4o-mini (78.7%) outperforms GPT baselines using the premium GPT-5.2 (71.7%) on Agent-A, demonstrating architecture matters more than model capability. Agent-B: DeepEval+ctx was not run on this agent (bar omitted).

5.2. Cost and Latency

Table 2 shows SS requires more LLM calls and wall-clock time than single-pass baselines, but achieves dramatically higher coverage. The cost-per-covered-intent is substantially lower for SS (gemini row): \$0.023 per covered ground-truth query versus \$0.034 for DeepEval (which covers only

Table 1. D1 ground-truth coverage (%) across all tools and both experiments. **Bold** = best per column. 95% bootstrap CIs (B=10,000) shown for key comparisons. Primary judge: gemini-2.5-flash.

	All-Gemini		All-GPT-4o-mini		gpt-5.2 Judge	
Tool	Agent-A	Agent-B	Agent-A	Agent-B	Agent-A	Agent-B
SS	**86.6%**	**94.3%**	**78.7%** [71.7, 85.8]	**90.6%** [84.9, 95.3]	**67.7%** [59.1, 75.6]	**85.8%** [79.2, 92.5]
GPT Structured (gem.)	71.7%	87.7%	26.8% [19.7, 34.6]	71.7%	—	—
GPT Simple (gem.)	65.4%	87.7%	30.7% [22.8, 38.6]	86.8%	24.4% [17.3, 32.3]	—
DeepEval+SS Ctx	—	—	50.8%	—	—	—
DeepEval evolved	15.7%	80.2%	27.6% [19.7, 35.4]	72.6%	18.1% [11.8, 25.2]	53.8%
DeepEval styled	13.4%	—	24.4% [17.3, 32.3]	71.7%	11.8%	58.5%

Table 2. Generation cost comparison (Agent-A).

Tool	Scen.	LLM Calls	Est. Cost	Time
SS (gemini)	228	∼80	∼$2.50	45m
SS (4o-mini)	345	∼100	∼$0.80	120m
DeepEval	250	∼30	∼$1.20	15m
GPT-Str (5.2)	47	1	∼$0.50	2m

27.6%).

5.3. D2 Cross-Tool Coverage

Table 3. D2 cross-tool matrix (Agent-A, gpt-4o-mini). Row X, Col Y = % of X covered by Y.

	SS	DE	GPT-S	GPT-St
SS (345)	100%	26.4%	65.8%	56.5%
DE (250)	94.4%	100%	50.8%	42.0%
GPT-S (231)	93.5%	37.7%	100%	91.3%
GPT-St (248)	90.7%	33.5%	92.3%	100%

Table 3 shows SS generates the most unique content—only 26.4% of SS scenarios are covered by DeepEval. SS subsumes 90–94% of every other tool's scenarios.

5.4. Intent Distribution and Collapse Analysis

Table 4. Intent distribution (Agent-A). DeepEval concentrates 98% on ride/booking.

Intent	GT %	DE %	SS %	GPT-S %
ride/booking	16%	**98%**	22%	35%
restaurant/food	32%	1%	28%	20%
flight	13%	0%	12%	10%
places	9%	0%	10%	8%
other/multi	26%	1%	24%	22%
weather+deliv.	4%	0%	4%	5%

Table 4 and Figure 3 confirm the root cause: DeepEval's distribution has near-zero entropy across intents, while SS closely mirrors the ground truth.

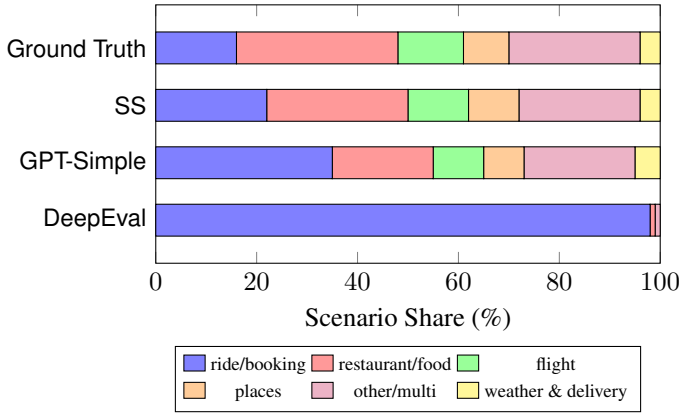

Figure 3. Intent distribution across tools (Agent-A). DeepEval concentrates 98% on ride/booking; SS mirrors the ground truth.

Three pieces of evidence confirm intent collapse is structural: (1) **Scale does not solve it**: increasing DeepEval from 70 to 250 scenarios improves coverage by only +11.9pp while maintaining 98% ride concentration. (2) **Evolution does not solve it**: adding EvolutionConfig improves by only +3.2pp. (3) **Better context helps but is insufficient**: DeepEval + SS Context reaches 50.8% (+35pp from context), but still trails SS by 35.8pp despite 1.1× more scenarios.

6. Analysis

6.1. Architecture Beats Model

SS with the ∼10× cheaper gpt-4o-mini still beats GPT-5.2 baselines (Table 5). Model downgrade degrades SS by only 7.9pp versus 35–47pp for GPT baselines—the PolicyGraph and CoverageTracker compensate for weaker model capabilities (Figure 4). We note this comparison places SS on a cost-sensitive model against GPT baselines on the premium GPT-5.2; it does not isolate orchestration from model quality at the *strongest* capability level, which would require running SS itself on GPT-5.2. We expect that configuration

Table 5. Architecture vs. model quality. SS (gemini-2.5-flash) vs GPT baselines (GPT-5.2)—both frontier-tier models.

Agent	Tool (model)	D1	Δ vs best
Agent-A	**SS (gemini)**	**86.6%**	**+14.9pp**
	SS (gpt-4o-mini)	78.7%	+7.0pp
	GPT-Str (GPT-5.2)	71.7%	—
Agent-B	**SS (gemini)**	**94.3%**	**+6.6pp**
	SS (gpt-4o-mini)	90.6%	+2.9pp
	GPT-Str (GPT-5.2)	87.7%	—

to further widen SS's lead (a stronger model with the same coverage steering), but we did not run it here—SS makes many more LLM calls per run than the single-pass baselines, so a full SS sweep on the premium GPT-5.2 was beyond the compute and time budget available for this work—and we flag it as future work (§7) rather than asserting it.

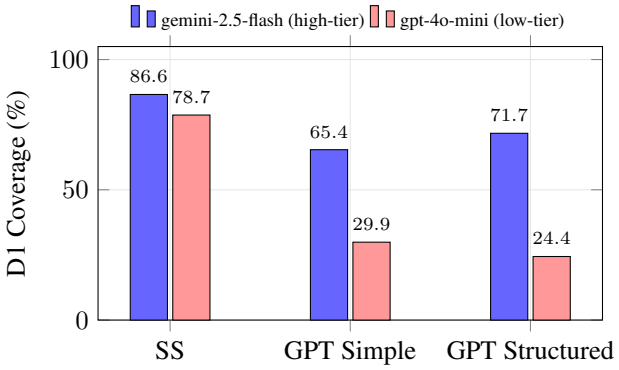

Figure 4. Model sensitivity on Agent-A. SS drops only 7.9pp when switching from gemini to gpt-4o-mini; GPT baselines collapse by 35–47pp.

6.2. Dual-Judge Agreement

Table 6. Judge agreement: gemini-2.5-flash vs. gpt-5.2 on D1.

Tool	Agree	κ
Agent-A / DeepEval	89.8%	0.571
Agent-A / GPT-Sim	95.3%	0.822
Agent-A / SS	89.0%	0.585

Table 6 shows 89.0–95.3% per-item agreement with Cohen's κ from 0.571 (moderate) to 0.822 (substantial). gpt-5.2 is consistently stricter but the **tool ranking is identical**: SS $\gg$ GPT $>$ DeepEval.

Self-preference and evaluation circularity. In the All-Gemini experiment, gemini-2.5-flash is both a generation model and the primary judge, which raises a self-preference concern: the judge might favor scenarios written in its own style and inflate SS's measured coverage. Three factors mitigate this. First, the metric is *not* a preference judgment over generated text but a match against a fixed, human-authored GT set that predates SS (§4.1); a tool is credited only when it independently reproduces a held-out intent, so a stylistic self-preference cannot manufacture coverage of an uncovered intent. Second, the cross-family gpt-5.2 judge—a different model lineage that did not generate any scenarios—yields an *identical* tool ranking and the same statistically significant separation (§6.3), so the ranking is not a gemini-specific artifact. Third, the largest gaps (e.g., +51pp over DeepEval with the cost-sensitive model) far exceed the 5–13pp judge-to-judge disagreement, so plausible judge bias cannot account for the effect. A residual circularity remains in that both the PolicyGraph and the coverage judgments are LLM-mediated; human validation of the semantic coverage judgments is the natural next step (§7).

6.3. Statistical Significance

We compute 95% bootstrap confidence intervals (B=10,000 resamples) over per-query binary coverage judgments. Table 7 summarizes the key comparisons for Agent-A.

Table 7. Bootstrap 95% CIs for Agent-A D1 coverage. Intervals are non-overlapping between SS and all baselines under both judges, confirming statistical significance ($p < 0.05$).

Judge	Tool	D1 (%)	95% CI
gemini	**SS**	78.7%	[71.7, 85.8]
	GPT Sim (gem.)	30.7%	[22.8, 38.6]
	DE evolved	27.6%	[19.7, 35.4]
	DE styled	24.4%	[17.3, 32.3]
gpt-5.2	**SS**	67.7%	[59.1, 75.6]
	GPT Sim	24.4%	[17.3, 32.3]
	DE evolved	18.1%	[11.8, 25.2]

For Agent-A, SS's CI lower bound exceeds every baseline's CI upper bound under both judges: the gap between SS $[71.7, 85.8]$ and the next-best baseline $[22.8, 38.6]$ is 33.1pp at the closest points. Under the stricter gpt-5.2 judge, the separation widens to 26.8pp. These non-overlapping intervals correspond to $p < 0.05$ by the bootstrap percentile method.

For Agent-B (narrow agent), CIs partially overlap—consistent with the expectation that coverage-aware orchestration provides the greatest advantage on diverse agents. Even so, SS achieves the highest point estimate under every condition (Table 1).

6.4. Red Teaming via Pluggable Backends

SS generates adversarial scenarios through a hybrid backend combining SS-native generators, DeepTeam, and Prompt-

Foo. The hybrid approach covers 12+ attack categories mapped to the OWASP LLM Top 10 (20) framework:

Table 8. Adversarial scenario generation (hybrid backend).

Agent	Red-team scen.	OWASP categories
Agent-A	149 (39% of pack)	5/10
Agent-B	320 (18% of pack)	5/10

Attack types include prompt injection, encoding attacks, authority escalation, PII extraction, gaslighting, hallucination probes, and tool/config extraction. Table 8 summarizes the coverage. Each backend contributes unique attack patterns: DeepTeam specializes in multi-turn social engineering, PromptFoo in structured jailbreaks, and SS-native generators in domain-specific adversarial scenarios. All backends benefit from SS's compiled context, producing more targeted attacks than standalone execution.

OWASP coverage analysis. The 5/10 OWASP coverage reflects the categories testable via conversational scenario generation. The covered categories—LLM01 (prompt injection), LLM03 (training data poisoning), LLM06 (sensitive information disclosure), LLM08 (excessive agency), and LLM09 (misinformation)—are the categories most relevant to conversational agent evaluation, where the attack surface is the user–agent dialogue. The five uncovered categories fall into two groups. **Structurally out of scope** (3): LLM02 (insecure output handling—requires runtime instrumentation of downstream consumers, not input scenarios), LLM05 (supply chain vulnerabilities—infrastructure-level concern about compromised models/libraries), and LLM10 (model theft—about extracting model weights, not testable via conversation). **Generators exist but unmapped in reporting** (2): LLM04 (model denial of service—SS includes `context_overflow` generators) and LLM07 (insecure plugin design—SS includes `tool_abuse` generators). These generators produce relevant adversarial scenarios but the reporting layer does not yet attribute them to OWASP IDs—a reporting gap, not a capability gap. Extending the tag mapping would raise coverage to 7/10, with the remaining 3 categories requiring infrastructure-level or runtime testing outside the scope of scenario generation.

6.5. Scenario Quality

LLM-judged quality scoring (clarity, self-containment, realism) shows SS matches GPT baselines (0.95 vs 0.95 overall), while DeepEval trails at 0.89–0.91. Higher coverage does not come at the expense of scenario quality.

7. Discussion & Future Work

Limitations. SS requires a DomainSpec configuration describing agent capabilities. We mitigate the authoring cost with a `--discover-domain` CLI flag that auto-generates an initial DomainSpec from the available sources (agent code, docs, seed queries) using an LLM; the user then reviews and refines it rather than writing it from scratch. The PolicyGraph quality still depends on source access and the richness of the provided context.

Generalization. Our evaluation covers two production conversational agents; while they bracket a broad and a narrow intent space, they are both dialogue agents from a single organization, and the magnitude of improvement may differ for other agent classes such as web-navigation, coding, or multimodal assistants. The intent-collapse mechanism is not specific to our setting, but quantifying SS on public conversational-agent benchmarks (and beyond dialogue agents) is important future work; the agents and GT sets here are proprietary and not publicly released.

Evaluation methodology. Both the PolicyGraph and the coverage judgments are LLM-mediated, so the metric inherits the ambiguity of semantic LLM-as-judge scoring. We mitigate this with a fixed human-authored GT target and a cross-family dual-judge study (§6.3), but we do not yet include a *human* evaluation of the coverage judgments; a human-agreement study against the LLM judges is the most direct way to close the residual circularity and is again a future work.

Isolating architecture from model. Our "architecture beats model" evidence compares SS on a cost-sensitive model against GPT baselines on premium GPT-5.2. We did not run SS *on* GPT-5.2—a full SS sweep is many times more LLM-call-intensive than a single-pass baseline, which put it beyond our compute and time budget here—and that run would most cleanly separate orchestration from model quality at the strongest capability level; we expect it to further widen SS's lead, but we mark it as future work rather than claiming it.

Red teaming. The red-teaming results report attack-category and OWASP coverage but not attack *success* rates or downstream scenario effectiveness; measuring whether generated adversarial scenarios actually elicit policy violations (and at what rate) is a separate evaluation we leave to future work.

Theoretical framing. The coupon-collector argument (§3.1) is intended as intuition for why single-pass generation is sample-inefficient, not as a tight formal bound; a rigorous analysis under realistic generation models is open.

Coverage report and gap analysis. Beyond raw coverage numbers, SS emits a structured *coverage report* at the end of every run. The report maps every node of the PolicyGraph

to (i) the number of scenarios targeting it, (ii) which of those scenarios survived the quality gate and consolidation, and (iii) whether any ground-truth query mapped to that node was matched. Nodes that remain uncovered after the configured number of rounds are surfaced as a *gap list*—often concentrated in regions where the source context is thin (e.g., edge-case policies only described in an unreferenced runbook). In practice this turns coverage into an actionable signal: developers respond by enriching the source context—adding the missing runbook, docstring, or tracked ticket—and re-running SS, rather than hand-authoring missing scenarios. We found this closed-loop workflow to be the primary mechanism by which intent coverage reached 86.6% on Agent-A without per-intent manual curation.

Multimodal extension. We are extending SS to support multimodal scenarios (text + image + audio). The design introduces structured presentation traits—`VoicePersona` for audio characteristics (tone, speed, accent) and `ImageConditions` for visual quality (blur, lighting, orientation)—that flow through to the simulator via metadata. This separates content traits (what the user says/shows) from presentation traits (how it sounds/looks), enabling systematic robustness testing across degraded conditions.

Broader impact. The intent collapse problem is not specific to our setting—any tool that generates test data from documents without modeling the target space will over-index on prominent topics. Coverage-aware orchestration is a necessary architectural principle for reliable synthetic data generation.

8. Conclusion

We presented Scenario Studio, a coverage-aware generation architecture for conversational AI testing. By modeling the agent's behavior space as a PolicyGraph and steering generation toward untested policies, SS achieves 86.6% ground-truth coverage versus 15.7% for the best baseline—a 5.5× improvement. Our dual-judge study confirms these gains are statistically significant, and our cross-model experiments demonstrate that architectural choices dominate model capability. We believe coverage-aware orchestration is a necessary principle for reliable synthetic test data generation.

Acknowledgements

We thank our colleagues at Uber and the partner agent teams whose production systems and evaluation suites made this study possible, and the ICML *Agents in the Wild* reviewers for feedback that improved the paper's treatment of ground-truth construction, judge bias, and PolicyGraph quality.

Impact Statement

This work aims to improve the reliability of testing for production conversational AI agents by increasing the coverage and realism of synthetic test scenarios, which can surface under-tested capabilities and safety-relevant failures before deployment. The same coverage-aware machinery is dual-use—it also drives red-teaming generators—so we restrict released artifacts to methodology and aggregate metrics and do not release the proprietary agents, ground-truth sets, or adversarial scenario packs. We see no immediate negative societal impacts beyond those general to synthetic-data and red-teaming tooling.

References

[1] Confident AI. DeepEval: The open-source LLM evaluation framework. https://github.com/confident-ai/deepeval, 2024.

[2] O. Khattab et al. DSPy: Compiling declarative language model calls into self-improving pipelines. In *ICLR*, 2024.

[3] PromptFoo. PromptFoo: Test your LLM app. https://github.com/promptfoo/promptfoo, 2024.

[4] Y. Wang et al. Self-Instruct: Aligning language models with self-generated instructions. In *ACL*, 2023.

[5] N. Nayak et al. Learning to generate instruction tuning datasets for zero-shot task adaptation. In *ACL Findings*, 2024.

[6] P. Yu et al. CoT-Self-Instruct: Building high-quality synthetic prompts for reasoning and non-reasoning tasks. https://arxiv.org/abs/2507.23751, 2025.

[7] A. Zhu et al. BARE: Combining base and instruction-tuned language models for better synthetic data generation. https://arxiv.org/abs/2502.01697, 2025.

[8] Y. Liu et al. G-Eval: NLG evaluation using GPT-4 with better human alignment. In *EMNLP*, 2023.

[9] L. Zheng et al. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *NeurIPS*, 2023.

[10] X. Liu et al. AgentBench: Evaluating LLMs as agents. In *ICLR*, 2024.

[11] G. Mialon et al. GAIA: A benchmark for general AI assistants. In *ICLR*, 2024.

[12] P. Liang et al. Holistic evaluation of language models. *Transactions on Machine Learning Research*, 2023.

[13] C. Ma et al. AgentBoard: An analytical evaluation board of multi-turn LLM agents. In *ACL*, 2024.

[14] Y. Qin et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *ICLR*, 2024.

[15] Y. Shen et al. TaskBench: Benchmarking large language models for task automation. In *NeurIPS Datasets and Benchmarks*, 2024.

[16] K. Levy et al. IntellAgent: A multi-agent framework for evaluating conversational AI systems. https://arxiv.org/abs/2501.11067, 2025.

[17] M. Zalewski. American Fuzzy Lop (AFL). https://lcamtuf.coredump.cx/afl/, 2014.

[18] K. Claessen and J. Hughes. QuickCheck: A lightweight tool for random testing of Haskell programs. In *ICFP*, 2000.

[19] Confident AI. DeepTeam: Red teaming for LLM applications. https://docs.confident-ai.com/docs/red-teaming-introduction, 2024.

[20] OWASP Foundation. OWASP Top 10 for LLM Applications. https://owasp.org/www-project-top-10-for-large-language-model-applications/, 2025.