EDGEGEN: Improving Tool-Calling Agents Beyond Happy Paths with Synthetic Edge Case Generation

Anonymous ACL submission

Abstract

Tool-calling LLM agents are increasingly deployed in enterprise applications. However, effective evaluation and optimization require high-quality, diverse task datasets that are often difficult to obtain due to privacy and other constraints. Existing synthetic task generation methods often produce generic tasks that ignore an agent's underlying state or database and fail to reflect real-world usage diversity. We propose EDGEGEN, a synthetic task generation framework that extracts compliance rules from an agent's specification and uses them to generate database-grounded edge-case tasks designed to violate these rules. When combined with existing synthetic data generation techniques, EDGEGEN enables agent improvement through finetuning and harness optimization. The resulting pipeline forms a fully automated closed-loop system that requires no human annotation. Finetuning on data generated by EDGEGEN yields a consistent mean progress improvement of **+2%** to **+42%** on τ^2-bench airline domain, while other baseline methods show degradation for some models. On the other hand, for harness optimization, our method shows a mean progress improvement of **+10%** and **+30%** over the human-curated and base harnesses, respectively, for the Gemma-4-e4b model[1].

1 Introduction

Large Language Model (LLM) agents are rapidly becoming a core component of enterprise systems. These agents must autonomously interact with tools, query databases, and execute multi-step workflows while adhering to complex business policies and operational constraints (Schick et al., 2023; Patil et al., 2024; Qin et al., 2024). As these agents scale up and interact in more varied conditions, improving the reliability and policy compliance of such systems has become increasingly important. Whether through reinforcement learning (Feng et al., 2024; Chen et al., 2025), prompt engineering or harness optimization (Lee et al., 2026; Zhang et al., 2026a,b), processes ultimately depend on high quality task data.

However, obtaining such data remains a major challenge. Real production logs are often inaccessible due to privacy concerns, while manually creating realistic tasks the agent might encounter is expensive and difficult to scale. At the same time, prior works on synthetic data generation consistently identify *quality* and *diversity* as the key factors of downstream performance (Liu et al., 2024a; Chen et al., 2024). These limitations have motivated a growing body of work on synthetic task generation for tool-use LLM agents (Wang et al., 2023; Liu et al., 2025, 2024b; Prabhakar et al., 2026).

Despite the progress, existing approaches largely focus on generic function calling tasks and often fail to capture the operational constraints of the agents. In practice, correctness depends not only on whether the agent can invoke the right tool, but also whether it respects domain-specific business rules. For example, an airline-support agent should deny refund requests that violate eligibility constraints, apply baggage policies conditioned on loyalty tiers, etc. These are inherently stateful and policy-dependent behaviors that generic synthetic tasks rarely evaluate.

Coverage is another dimension that is equally important. Agents that perform well on narrow or templated evaluations may still fail in deployment when faced with rare edge cases (Zhang et al., 2024; Ruan et al., 2024; Zhou et al., 2024). Robust evaluation therefore requires diverse, grounded scenarios that systematically probe an agent's failure modes rather than testing common "happy-path" interactions.

To address this gap, we introduce EDGEGEN,

[1] Code and dataset are available in the repository https://github.com/OfflineSdg/EdgeGen.

a framework for generating diverse, database-grounded tasks that explicitly stress-test business-rule compliance. Our pipeline in Figure 1 first extracts these rules from an agent's policy document. It then constructs violation scenarios by enumerating combinations of these rules, grounds each scenario in database states using a scenario-aware SQL agent which samples values that make the task feasible, and verifies the validity of every generated task before admission. The resulting dataset spans a broad spectrum of difficulty, ranging from simple single tool interaction to complex multi-step workflows involving multiple policy violations.

In our setting, we pair EDGEGEN with a synthetic database generation method (Anonymous, 2026)[2] that produces a relational database that inherently encodes the business rules. Together, these components form a fully synthetic closed loop agent improvement pipeline. The synthetic database provides the environment state, EDGEGEN generates grounded tasks against that state, and the resulting trajectories are used for finetuning or harness optimization. The entire process operates without production logs, human annotated tasks or real data, yet still improves performance on human-curated test sets.

Empirically, using supervised-finetuning and harness optimization, we show that, EDGEGEN consistently improves performance across multiple benchmarks, model families and against different synthetic generation baselines. On τ^2-bench airline (Barres et al., 2025) and ToolSandbox (Lu et al., 2025) it yields a substantial gain for small and mid-sized models, showing a $+42\%$ improvement on average progress for Qwen 2.5-3b relative to the base version. For harness optimization, our method shows a relative mean progress improvement of $+10\%$ and $+30\%$ over the human-curated and base harnesses, respectively, on τ^2-bench airline domain using the Gemma-4-e4b model.

Contributions

1. We propose EDGEGEN, a violation-driven synthetic task generation framework that extracts compliance rules from agent specifications, enumerates structured combinations of rule violations, and grounds tasks in executable database states.

2. We introduce a fully automated closed-loop system that combines synthetic database generation, EDGEGEN task construction, produce high-quality training and evaluation data without human annotations or production logs, enabling agent optimization.

3. We empirically demonstrate that explicitly covering edge cases improves tool-using agent performance across benchmarks and model scales.

2 Related Work

2.1 Synthetic Data and Task Generation

Self-instruct (Wang et al., 2023) establishes the recipe of bootstrapping instruction data from their own generations. Subsequent work Chen et al. (2024) has studied impact of synthetic data, including the role of diversity in downstream performance during pretraining and finetuning. APIGen (Liu et al., 2024b) proposes an automated pipeline for generating diverse and verifiable function-calling datasets from executable APIs, while APIGen-MT (Prabhakar et al., 2026) further extends this framework to multi-turn tool-use. ToolACE (Liu et al., 2025) further improves synthetic function-calling data generation via self-evolving synthesis and a complexity-controlled dialog generation with verification layer to ensure data quality.

In this work, we repurpose TaskBench (Shen et al., 2024), from which we adopt the tool graph sampling strategies (Node, Chain and Directed-Acyclic Graph (DAG)) for workflow construction. Similarly, we adapt FuncBenchGen (Maekawa et al., 2026) as described in Sec. 4. Both share a common limitation when applied to agent benchmarks that operate using a specific behavioral policy. For TaskBench (Shen et al., 2024), task descriptions are either generated using an LLM using back-instruct and sampled subgraphs. FuncBenchGen's (Maekawa et al., 2026) tasks are constructed from function schemas with programmatically assigned values. As a result, they are not systematically enumerating over a structured space of behavioral rule violations nor grounded in a database state. This leaves edge case scenarios underexplored. EDGEGEN addresses these shortcomings by first enumerating over the power-set of behavioral rules, and second, uses a scenario-aware SQL agent to fetch actual database state.

2.2 Agent Evaluation

[2]Concurrent anonymous submission, under review. Anonymized copy provided in supplementary material.

The τ-Bench (Yao et al., 2024) and τ^2-Bench (Barres et al., 2025), introduce realistic multi-turn customer-service scenarios, grounded in an airline or retail policy, acting as our primary evaluation benchmark. ToolSandbox (Lu et al., 2025) provides a tool device-management environment with stateful constraints and minimal system prompt, serving as our secondary benchmark. Other broader family of agent benchmark and evaluation approaches are AgentBench (Liu et al., 2023), WorkArena (Drouin et al., 2024), API-Bank (Li et al., 2023), MINT (Wang et al., 2024), and AgentBoard (Ma et al., 2024) which follow common evaluation paradigm for tool-use agents. While each provides a curated set of test cases, none provides a synthetic-data layer for automatically expanding existing test cases evaluation to support agent improvement. EDGEGEN closes this gap by introducing a fully automated closed-loop system: from simulating compliance-aware databases and generating grounded training and evaluation tasks, to optimizing agents based on the synthetic data.

3 Methodology

3.1 Problem Formulation

A deployment of a tool-calling agent can be characterized by three artefacts that are typically available during development. The first is the set of tools accessible to the agent, denoted as:

$$T = \{t_1, \ldots, t_n\},$$

where each tool t_i is associated with input $\text{In}(t_i)$ and output $\text{Out}(t_i)$ signatures that define its data types. The second artefact is the policy document P, which specifies the behavioral constraints and operational requirements the agent must follow. The third is the relational database D that the agent interacts with during execution through reads and writes.

The policy document P encodes behavioral rules in natural language. We formalize these rules as a set of atomic compliance constraints implied by P:

$$R = \{r_1, \ldots, r_n\}.$$

Each rule r_i corresponds to a single verifiable assertion about the agent's behavior. For example, a rule may specify that "the agent must verify the user identity before modifying a reservation." Our framework extracts these rules automatically from P.

We define a test case as a four-element tuple:

$$x = \langle q, w, A, S \rangle,$$

where q denotes a user task description grounded in concrete values sampled from the database. The workflow:

$$w = \{t_i, \ldots, t_k\},$$

represents an ordered sequence of k tool invocations drawn from T. The third element,

$$A = \{a_1, \ldots, a_n\},$$

is the set of natural language assertions (Chong et al., 2026) associated with the task. These assertions serve as evaluation targets and are used to determine whether the agent successfully completed the task. Finally, $S \subseteq R$ denotes the set of policy rules that the test case intends to violate or stress-test.

A test case x is considered valid with respect to the database D when two conditions are satisfied. First, every concrete entity referenced in the task description q and workflow w must exist in the database (data grounding). Second, the workflow w must execute successfully against D without violating any tool preconditions or database constraints (scenario feasibility). These validity requirements address two common failure modes observed in prior synthetic task generation methods (Shen et al., 2024; Maekawa et al., 2026): hallucinated identifiers and infeasible tool sequences whose required database state does not exist. Refer to Appendix-A.3 for some examples.

3.2 Test Case Generation Pipeline

Based on this problem formulation, EDGEGEN generates valid, database-grounded test cases through a multi-stage pipeline that progressively constructs workflows, enumerates policy violations, and grounds the resulting scenario in feasible database states.

Stage 1: Tool graph construction and path sampling. Inspired by Shen et al. (2024), we construct a directed dependency graph $G = (V, E)$ where $V = T$ and an edge exists between two tools if their data types are compatible:

$$(t_i, t_j) \in E \iff \text{Out}(t_i) \cap \text{In}(t_j) \neq \emptyset.$$

This construction follows the resource type formulation mentioned in (Shen et al., 2024). However,

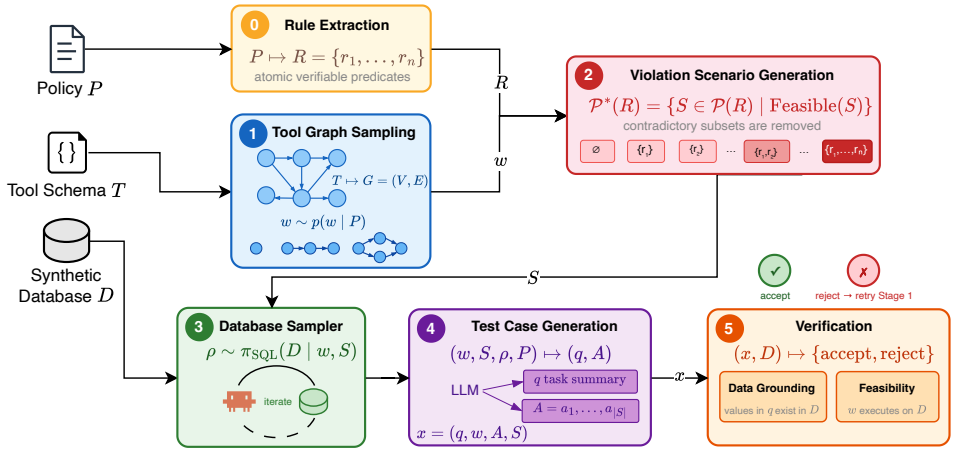

Figure 1: Illustrates the automated EDGEGEN pipeline for generating synthetic, database-grounded test cases x designed to violate business policies. The system first extracts compliance rules R from an agent's policy document and enumerates combinations of these rules $\mathcal{P}^*(R)$ to create structured violation scenarios S. It then samples a tool workflow w, grounds the scenario in a database D state using a SQL agent π_{SQL}, and verifies the final task for data grounding and scenario feasibility.

we identified that sampling a feasible workflow should also depend on policy P. Most of these agent prompts or policies mention constraints about which tool to call first before proceeding with an action. These rules are often too complex to explicitly represent them in code. Therefore, we define a distribution over valid workflow structure induced by P. We use an LLM to select a valid set conditioned on P. A workflow w is sampled as:

$$w \sim p(w \mid P).$$

From this we sample a NODE, CHAIN, or DAG, which correspond to a single-tool execution, linear compositions, or branching multi-tool workflows, respectively.

Stage 2: Violation scenario generation. Given the extracted rule set R, we define the space of all possible compliance conditions as its power set:

$$S \subseteq R, \quad S \in \mathcal{P}(R).$$

where $\mathcal{P}(R)$ specifies the power set of all the extracted rules. Each subset S represents a set of simultaneously enforced or violation constraints. The size of the violation set, $|S|$, controls the complexity of the generated scenario. When $|S| = 0$, the task corresponds to a standard "happy-path" interaction. When $|S| = 1$, the test case isolates a single policy constraint. Larger violation sets ($|S| \geq 2$) generate more challenging scenarios that require the agent to simultaneously enforce multiple interacting rules.

However, not all subsets are jointly satisfiable. Therefore, we define a feasibility filter:

$$\mathcal{P}^*(R) = \{S \in \mathcal{P}(R) \mid \text{Feasible}(S)\},$$

where $\text{Feasible}(S)$ is implemented using an LLM-based consistency checker that removes logically incompatible rule combinations.

Stage 3: Scenario-aware database sampling. Given a workflow $w \sim p(w \mid P)$ and a violation set $S \in \mathcal{P}^*(R)$, we must instantiate a concrete database state that satisfies both the execution structure and the constraint configuration.

We define a grounding procedure implemented by an iterative SQL agent:

$$\rho \sim \pi_{\text{SQL}}(D \mid w, S),$$

where ρ is a set of database rows that instantiate the scenario. If no valid ρ is found, the system resamples w and S.

Stage 4: Test case generation. Given an agent's specifications P, sampled workflow $w \sim p(w \mid P)$, a violation set $S \in \mathcal{P}^*(R)$, and a grounded database state $\rho \subset D$, we generate the task summary q and natural language assertions A using an LLM. The task summary will contain concrete entities present in the database (eg: identifiers, timestamps, etc), and the natural language assertions will encode the expected agent response under the given constraints (eg: denial of an operation due to policy violation).

Stage 5: Verification. Since both workflow sampling and LLM-based generation may introduce inconsistencies, we apply a verification step that ensures logical consistency with the database state D. We define a verifier, which will perform two independent checks, the first is data grounding, where every entity, identity, and attribute referenced in q and in the arguments of w must exist in D. This is validated by the SQL agent, which will execute deterministic SQL queries over the database schema. The second is the feasibility of the scenario, where we check that the sampled workflow must be executable against D. This ensures that each tool invocation in w is valid given the current state transitions induced by prior steps. A test case x is accepted only if both conditions hold. Otherwise, it is discarded, and the pipeline restarts from Stage 1 with a newly sampled workflow and violation set.

3.3 Complexity-Aware Generation

The generation process has two orthogonal axes of controllable complexity. The first axis controls the structural complexity via the workflow distribution. Increasing the sampling rate for CHAIN or DAG increases the complexity of the task. The second axis is scenario complexity as determined by $|S|$. Together, these two axes define a structured coverage space over task difficulty. Sampling uniformly or non-uniformly over this grid induces a controllable distribution over scenario complexity, spanning simple database lookups to multi-step, multi-policy decision workflows. We empirically analyze the effect of finetuning with different scenario complexities in Section 4.5.

4 Experiments

4.1 Experimental Setup

We evaluate EDGEGEN on τ^2-bench airline and retail domains (Barres et al., 2025), and ToolSandbox (Lu et al., 2025) dataset. τ^2-bench airline is a customer-service benchmark with 14 tools, from which we use 10 human-curated samples for evaluation. ToolSandbox focuses on device-management workflows and contains 33 tools; we select 15 human-curated samples. τ^2-bench retail is another customer-service benchmark with 15 tools, for which we use 10 human-curated samples. All test sets are drawn from the original author-provided samples.

We generate 15 diverse tasks using EDGEGEN for all the benchmarks. For finetuning, we collect the expert demonstrations from gpt-5.4 as agent model, over 8 trials for each sample, thus giving us 120 rollouts. We then retain only the successful traces for supervised-finetuning. We evaluate six open-source, instruction-tuned models spanning multiple scales and architecture: Qwen 2.5-3b, Qwen 3.5-4b, Qwen 3.5-9b, Qwen 3.5-35b-A3b (Yang et al., 2024; Team, 2024; Qwen Team, 2026), Gemma 4-e2b, and Gemma 4-e4b. All models are evaluated on the same held-out test sets provided by the authors alof the benchmarks.

For agent harness optimization on τ^2-bench airline domain, we use 10 scenarios (either human curated or synthetic EDGEGEN samples) for train, 8 synthetic EDGEGEN scenarios for validation, and 10 original, human-curated samples for test. For the test set, it is the same as that used in the finetuning experiments. We conduct this set of experiments with Gemma 4-e4b and gpt-5.4 as the underlying agent model. For the Gemma 4-e4b model, which has lower baseline performance than gpt-5.4, we allocate a higher budget of 8 evolution iterations, compared to 4 iterations for gpt-5.4, due to gpt-5.4's stronger initial performance. In all experiments, the harness is optimized on the train split, and the best candidate is selected based on the performance on the validation split. During the evolution process, each candidate is evaluated using two evaluation trials. For reporting on the human-curated test split derived from the original dataset, we use 8 trials per scenario.

In all our experiments, we use the Talk, Evaluate, Diagnose (TED) framework (Chong et al., 2026) which has a user simulator and LLM-as-a-judge to evaluate the models over 8 trials per scenario. We use the gpt-4.1 model for the expert persona user simulator and the LLM-as-a-judge. We report the mean progress rate, which measures average progress across all trials, and maximum progress rate, which measures best-case performance by taking the maximum progress across trials. For the agent harness experiments, we also report the efficiency of the agent in terms of average number of tokens and the average number of tool calls made.

4.2 Baselines

For the finetuning experiments, we compare EDGEGEN against 5 baselines. The base model baseline uses the original instruction-tuned checkpoint without additional finetuning. Human-curated baseline finetunes the model on the original benchmark trajectories provided by the dataset creators. The

Model	Base	Human-Curated	Naive	FuncBenchGen	TaskBench	EdgeGen (Ours)
			τ^2-bench Airline Domain			
Qwen2.5-3b	0.35 | 0.26	**0.39**$_{\uparrow 0.04}$ | 0.30$_{\uparrow 0.04}$	0.34$_{\downarrow 0.01}$ | 0.29$_{\uparrow 0.03}$	**0.39**$_{\uparrow 0.04}$ | 0.34$_{\uparrow 0.08}$	0.28$_{\downarrow 0.07}$ | 0.25$_{\downarrow 0.01}$	**0.39**$_{\uparrow 0.04}$ | **0.37**$_{\uparrow 0.11}$
Qwen3.5-4b	0.89 | 0.76	0.90$_{\uparrow 0.01}$ | 0.79$_{\uparrow 0.03}$	**0.92**$_{\uparrow 0.03}$ | 0.79$_{\uparrow 0.03}$	0.85$_{\downarrow 0.04}$ | 0.72$_{\downarrow 0.04}$	0.87$_{\downarrow 0.02}$ | 0.74$_{\downarrow 0.02}$	**0.92**$_{\uparrow 0.03}$ | **0.79**$_{\uparrow 0.03}$
Qwen3.5-9b	0.92 | 0.80	0.87$_{\downarrow 0.05}$ | 0.78$_{\downarrow 0.02}$	0.92$_{\pm 0.00}$ | 0.81$_{\uparrow 0.01}$	0.92$_{\pm 0.00}$ | 0.78$_{\downarrow 0.02}$	0.89$_{\downarrow 0.03}$ | 0.78$_{\downarrow 0.01}$	**0.95**$_{\uparrow 0.03}$ | **0.85**$_{\uparrow 0.05}$
Qwen3.5-35b	**0.92** | 0.81	0.88$_{\downarrow 0.04}$ | 0.76$_{\downarrow 0.05}$	0.84$_{\downarrow 0.08}$ | 0.73$_{\downarrow 0.08}$	**0.92**$_{\pm 0.00}$ | 0.79$_{\downarrow 0.02}$	0.85$_{\downarrow 0.07}$ | 0.76$_{\downarrow 0.05}$	**0.92**$_{\pm 0.00}$ | **0.82**$_{\uparrow 0.01}$
Gemma-4-e4b	0.66 | 0.43	0.69$_{\uparrow 0.03}$ | 0.50$_{\uparrow 0.07}$	0.70$_{\uparrow 0.04}$ | 0.48$_{\uparrow 0.05}$	0.69$_{\uparrow 0.03}$ | 0.51$_{\uparrow 0.08}$	0.70$_{\uparrow 0.04}$ | 0.55$_{\uparrow 0.12}$	**0.79**$_{\uparrow 0.13}$ | **0.56**$_{\uparrow 0.13}$
Gemma-4-e2b	0.47 | 0.38	0.39$_{\downarrow 0.08}$ | 0.33$_{\downarrow 0.05}$	0.47$_{\pm 0.00}$ | 0.40$_{\uparrow 0.02}$	0.49$_{\uparrow 0.02}$ | 0.41$_{\uparrow 0.03}$	**0.56**$_{\uparrow 0.09}$ | 0.39$_{\uparrow 0.01}$	0.55$_{\uparrow 0.08}$ | **0.42**$_{\uparrow 0.04}$
			ToolSandbox			
Qwen2.5-3b	0.84 | 0.60	0.81$_{\downarrow 0.03}$ | 0.64$_{\uparrow 0.04}$	0.83$_{\downarrow 0.01}$ | 0.61$_{\uparrow 0.01}$	N/A	0.81$_{\downarrow 0.03}$ | 0.60$_{\pm 0.00}$	**0.86**$_{\uparrow 0.02}$ | **0.67**$_{\uparrow 0.07}$
Qwen3.5-4b	0.66 | 0.55	0.75$_{\uparrow 0.09}$ | **0.68**$_{\uparrow 0.13}$	0.78$_{\uparrow 0.12}$ | 0.67$_{\uparrow 0.12}$	N/A	**0.78**$_{\uparrow 0.12}$ | 0.63$_{\uparrow 0.08}$	0.77$_{\uparrow 0.11}$ | 0.66$_{\uparrow 0.11}$
Qwen3.5-9b	0.73 | 0.60	0.78$_{\uparrow 0.05}$ | 0.68$_{\uparrow 0.08}$	0.78$_{\uparrow 0.05}$ | 0.68$_{\uparrow 0.08}$	N/A	**0.83**$_{\uparrow 0.10}$ | 0.69$_{\uparrow 0.09}$	0.82$_{\uparrow 0.09}$ | **0.71**$_{\uparrow 0.11}$
Qwen3.5-35b	0.70 | 0.54	**0.81**$_{\uparrow 0.11}$ | 0.65$_{\uparrow 0.11}$	0.76$_{\uparrow 0.06}$ | 0.59$_{\uparrow 0.05}$	N/A	0.73$_{\uparrow 0.03}$ | 0.63$_{\uparrow 0.09}$	**0.81**$_{\uparrow 0.11}$ | **0.65**$_{\uparrow 0.11}$
Gemma-4-e4b	0.82 | 0.64	0.88$_{\uparrow 0.06}$ | 0.76$_{\uparrow 0.12}$	**0.92**$_{\uparrow 0.10}$ | 0.77$_{\uparrow 0.13}$	N/A	0.88$_{\uparrow 0.06}$ | 0.74$_{\uparrow 0.10}$	0.90$_{\uparrow 0.08}$ | **0.77**$_{\uparrow 0.13}$
Gemma-4-e2b	0.65 | 0.53	0.78$_{\uparrow 0.13}$ | **0.65**$_{\uparrow 0.12}$	0.73$_{\uparrow 0.08}$ | 0.62$_{\uparrow 0.09}$	N/A	0.78$_{\uparrow 0.13}$ | 0.63$_{\uparrow 0.10}$	**0.85**$_{\uparrow 0.20}$ | **0.65**$_{\uparrow 0.12}$
			τ^2-bench Retail Domain			
Qwen2.5-3b	0.61 | 0.30	0.68$_{\uparrow 0.07}$ | 0.30$_{\pm 0.00}$	0.67$_{\uparrow 0.06}$ | 0.31$_{\uparrow 0.01}$	0.66$_{\uparrow 0.05}$ | 0.34$_{\uparrow 0.04}$	0.74$_{\uparrow 0.13}$ | 0.33$_{\uparrow 0.03}$	**0.78**$_{\uparrow 0.17}$ | **0.40**$_{\uparrow 0.10}$
Qwen3.5-4b	0.92 | 0.81	0.92$_{\pm 0.00}$ | **0.87**$_{\uparrow 0.06}$	0.92$_{\pm 0.00}$ | 0.85$_{\uparrow 0.04}$	**0.94**$_{\uparrow 0.02}$ | 0.84$_{\uparrow 0.03}$	0.92$_{\pm 0.00}$ | 0.80$_{\downarrow 0.01}$	0.92$_{\pm 0.00}$ | 0.84$_{\uparrow 0.03}$
Qwen3.5-9b	0.92 | 0.84	0.92$_{\pm 0.00}$ | 0.81$_{\downarrow 0.03}$	**0.99**$_{\uparrow 0.06}$ | 0.85$_{\uparrow 0.01}$	0.92$_{\pm 0.00}$ | 0.80$_{\downarrow 0.04}$	0.92$_{\pm 0.00}$ | 0.87$_{\uparrow 0.03}$	0.92$_{\pm 0.00}$ | **0.90**$_{\uparrow 0.06}$
Qwen3.5-35b	0.99 | 0.90	0.99$_{\pm 0.00}$ | 0.88$_{\downarrow 0.02}$	0.99$_{\pm 0.00}$ | 0.86$_{\downarrow 0.04}$	0.99$_{\pm 0.00}$ | 0.87$_{\downarrow 0.03}$	**1.00**$_{\uparrow 0.01}$ | **0.92**$_{\uparrow 0.02}$	0.99$_{\pm 0.00}$ | 0.92$_{\uparrow 0.02}$
Gemma-4-e4b	0.90 | 0.46	0.86$_{\downarrow 0.04}$ | 0.54$_{\uparrow 0.08}$	0.88$_{\downarrow 0.02}$ | 0.62$_{\uparrow 0.16}$	0.85$_{\downarrow 0.05}$ | 0.49$_{\uparrow 0.03}$	0.92$_{\uparrow 0.02}$ | **0.63**$_{\uparrow 0.17}$	**0.95**$_{\uparrow 0.05}$ | 0.60$_{\uparrow 0.14}$
Gemma-4-e2b	0.89 | 0.64	0.87$_{\downarrow 0.02}$ | 0.62$_{\downarrow 0.02}$	**0.94**$_{\uparrow 0.05}$ | 0.64$_{\pm 0.00}$	0.87$_{\downarrow 0.02}$ | 0.58$_{\downarrow 0.06}$	0.86$_{\downarrow 0.02}$ | 0.60$_{\downarrow 0.04}$	0.94$_{\uparrow 0.05}$ | **0.67**$_{\uparrow 0.03}$

Table 1: Progress rate results across τ^2-bench Airline (top), ToolSandbox (middle), and τ^2-bench Retail (bottom) benchmarks for all models and different training datasets. Each cell reports the average maximum progress (left) and average mean progress (right). Absolute percentage point changes over the base (untuned) model are colored blue (improvement) or red (degradation). N/A indicates that the data generation method did not produce enough valid training trajectories to perform finetuning.

Naive baseline concatenates tool schemas, agent specifications and sampled database contents into a single prompt and asks an LLM to generate tasks directly, without any tool samples or grounding verification.

For Taskbench (Shen et al., 2024), we adapt the codebase to utilize our SQL Agent which samples targeted database state for sampled workflow. We adapt TaskBench because its out-of-the-box implementation does not generate tasks with real database values, and the method is not directly compatible with our benchmarks. Similarly, for FuncBenchGen (Maekawa et al., 2026), we adapt the original pipeline to our benchmarks' environments by augmenting the tool schema with explicit I/O annotations and replacing value sampling with domain-specific database retrieval.

For the agent harness experiments, we report test-set performance across three settings: the base agent harness, the harness optimized using EDGEGEN train test cases, and the harness optimized using the train split derived from the provided human curated test cases in the τ^2-bench airline domain. To optimize the agent harness, we implemented a variant of the approach in Lee et al. (2026), where we equip the skill of the proposer agent to generate candidate harnesses via agent instruction refinement, tool description refinement, code mutation, or a combination of these. At every evolution iteration, the proposer agent proposes three candidates based on the train evaluation logs it sees.

4.3 Finetuning Results

Table 1 shows the change in mean progress relative to the base model. We observe that on τ^2-bench airline, EDGEGEN achieves a relative improvement of $+42.2\%$ (0.258 vs 0.366) on Qwen-2.5-3b. On ToolSandbox, the relative percentage improvement over base ranges from $+10.9\%$ (0.605 vs 0.671) to $+23.4\%$ (0.528 vs 0.652). ToolSandbox mostly focuses on tool-calling mechanisms, with fewer turns to complete the task. Therefore, we see most baselines improve the performance. In contrast to ToolSandbox, τ^2-bench airline domain has a dense policy which specific baggage requirements, membership level, etc. Here, we see that EDGEGEN consistently improves for all models, whereas other baselines decrease the performance. Finetuning using human-curated data regresses on 3 out of 6 models (up to -0.05 points (-13.16%) on gemma-4-e2b), and TaskBench regresses on 4 out of 6 models. This supports the hypothesis that

tasks with less coverage (no violation scenarios) is insufficient and can be actively harmful.

The increased in performances are substantial for smaller models (Qwen 2.5-3b and Gemma models), but for stronger models or easier datasets that already saturate, we see smaller gains. For some models, finetuning with human-curated training data underperforms synthetic data methods under same training budget constraints. We hypothesize that the limited set of curated scenarios fails to adequately cover the combinatorial space of policy violations encountered during evaluation, causing models to overfit to narrow interaction patterns. In contrast, EDGEGEN explicitly maximizes coverage over combinations of compliance rules.

Although FuncBenchGen and TaskBench both incorporate tool-graph construction and, in our adaptation, database grounding, they still frequently reduce performance in the τ^2-bench airline domain. For ToolSandbox, FuncBenchGen produced only a single valid rollout, suggesting that the vast majority of generated tasks were infeasible or invalid.

4.4 Agent Harness Optimization

EDGEGEN-generated train samples can be used to finetune the underlying agent model. However, for black-box models where finetuning is not feasible, the same set of samples can still be leveraged to optimize the agent harness.

In Table 2, we compare the baseline harness test performance with the optimized harness using EDGEGEN samples and original human-curated train data. For experiments with agent model Gemma 4-e4b, we observed that the agent harness that is optimized with the synthetic EDGEGEN samples performs significantly better than the baseline harness and its counterpart optimized with the human-curated samples, particularly in terms of the mean progress rate metric. The EDGEGEN-optimized harness shows an improvement on the mean progress rate from 0.43 to 0.56, corresponding to a 30% relative improvement over the baseline harness and also achieves a 10% relative improvement in the mean progress rate over the harness optimized using the original human-curated data. For gpt-5.4 model which has significantly larger capacity, we observe that the EDGEGEN-harness performance is within a 2-4% relative difference of the counterpart optimized on human-curated samples across both mean and maximum progress rate metrics. Additionally EDGEGEN-harness is more efficient as it requires less number of tool calls and tokens when compared to its counterpart optimized on the human-curated dataset.

	Performance		Efficiency	
Harness	**Max Prog ↑**	**Mean Prog ↑**	**Tools ↓**	**Tokens ↓**
Gemma-4-e4b				
Base	0.6583	0.4301	**5.8**	**81.0k**
Opt. (Human-Curated)	0.8167	0.5083	8.8	121.0k
Opt. (EDGEGEN)	**0.8208**	**0.5608**	6.9	85.0k
gpt-5.4				
Base	0.9500	0.7130	**7.4**	**53.8k**
Opt. (Human-Curated)	**0.9750**	**0.8307**	8.9	78.8k
Opt. (EDGEGEN)	0.9542	0.7979	7.9	66.5k

Table 2: Harness optimization results on the test split derived from the human-curated test cases in the τ^2-bench airline domain. We report progress rate alongside the average number of tool calls and token usage. ↑ / ↓ indicate whether higher or lower is better.

These positive findings suggest that the test cases generated by EDGEGEN are a strong substitute for actual human-curated test cases. By optimizing the harness on edge test cases intentionally designed to evaluate agent adherence to business compliance rules, we ensure that the resulting harness is optimized for broader coverage beyond only "happy-path" evaluation which make up the majority of the human-curated datasets. To better understand these improvements, we further analyze how EDGEGEN test cases are used to improve the harness over the original data using Gemma-4-e4b results as an example case study. We focus in particular on the structure of the resulting system prompts under different training distributions.

Under the human-curated train split samples, harness optimization is dominated by "please do X" approve-branch workflows. As a result, after several evolutions, the system prompt becomes a long procedural checklist that hard-codes business rules case-by-case (e.g. "cancellation requires a 24-hour window, an airline cancellation, a business-class reservation, or insurance with a covered reason").

In contrast, EDGEGEN's violation-driven task generation focuses more on covering boundary and rejection scenarios, enabling the harness optimizer to effectively refine the agent's standard execution workflow in challenging cases. For example, these edge cases include compensation-ineligibility denials, partial-cancellation rejections, fabricated disruption claims, and membership-tier mismatches. Hence, the system prompt under this signal can be collapsed into a short *discover→verify→confirm→write* protocol that

Opt. (Human-Curated)	Opt. (EDGEGEN)
Cancellation requires: 24hr window OR airline cancelled OR business OR insured+covered reason. Flown flights → transfer to human.	*Before any write operation (book, modify, cancel), verify all policy requirements are met and confirm with the user once. For cancellations, check all four eligibility conditions in the policy.*

Figure 2: Extracted system prompt for the Gemma-4-e4b agent on τ^2-bench airline domain. The prompt optimized on human-curated samples encodes lengthy cancellation rules, whereas the EDGEGEN-optimized prompt adopts a generic verification protocol that is potentially more robust to diverse edge-case scenarios. Full prompts in Appendix A.5.

Training Data	Qwen2.5-3B	Qwen3.5-4B
Base	0.35	0.89
Happy Path	0.39 $_{\uparrow 0.05}$	0.88 $_{\downarrow 0.01}$
1 Violation	**0.67** $_{\uparrow 0.32}$	**0.92** $_{\uparrow 0.03}$
2 Violations	0.24 $_{\downarrow 0.10}$	0.90 $_{\uparrow 0.02}$
3 Violations	0.38 $_{\uparrow 0.03}$	0.90 $_{\uparrow 0.02}$

Table 3: Complexity ablation on maximum progress rate. Each row reports performance after finetuning on the corresponding training condition; subscripts denote the absolute change relative to the unfinetuned Base model.

references the policy document and tool descriptions rather than enumerating them inline within the prompt. This solution is more robust to diverse edge-case scenario and work well for smaller models that has shorter context window. An example of the optimized prompt in the harness is shown in Figure 2. We demonstrate a more detailed case study showing the effectiveness of our synthetic data in Appendix A.5.

4.5 Complexity Ablation

To study the effect of scenario complexity, we perform an ablation over the number of policy violations embedded in each synthetic training example. We consider four settings: (1) happy path, where the agent fulfills the request without any constraint violations, (2) 1-violation, (3) 2-violations, (4) 3-violations. Each setting produces an equal size finetuning dataset, and we finetune Qwen-3.5-4b and evaluate it on the same held-out test set for τ^2-bench.

Table 3 reports the average maximum progress rate. For Qwen3.5-4b, finetuning on happy path trajectories alone provides no improvement over the base model (-0.01), suggesting that straightforward demonstrations are insufficient. Introducing a single policy violation yields a better performance, while any additional violation does not improve the model, but still remains above the baseline.

For Qwen2.5-3b, the trend is more pronounced. A single violation produces an improvement of $+0.32$, indicating that violation-aware training is impactful for model, possibly teaching the model domain-specific knowledge that happy path cases cannot effectively capture. However, at 2 and 3 violations, the performance collapses. We attribute this to the tasks being too complex for small models. Complex tasks produce longer traces, more intricate tool calling patterns which exceed the model's capacity.

4.6 Human Study

To validate our evaluation pipeline, we conduct human studies on successful and failed EDGEGEN trajectories. The results show that most successful runs are correct, and most failures reflect genuine agent limitations rather than flaws in task generation or evaluation. Only a small fraction arise from ambiguous assertions, invalid tasks, or judge errors, indicating that EDGEGEN produces reliable and meaningful scenarios. More details are provided in Appendix A.6.

5 Conclusion

We propose EDGEGEN, a synthetic task generation framework that extracts compliance rules from agent specifications, enumerates their combinations, and grounds scenarios in executable database states without human annotation. Our results show that explicit coverage of agent behavioral policies is critical for data quality: while finetuning on human-curated or generic synthetic datasets can degrade performance, training on EDGEGEN-generated data consistently improves results across multiple model families. For black-box models where finetuning is infeasible, the generated tasks also enable harness optimization, yielding a $+30\%$ relative improvement over the baseline with fewer tool calls and tokens. These findings demonstrate EDGEGEN's effectiveness as a scalable framework for evaluation and optimization.

Limitations

EDGEGEN relies on a structured policy document to extract compliance rules; agents with sparse, ambiguous, or informal specifications may yield incomplete rule sets and reduced edge-case coverage. Additionally, several pipeline stages depend on LLM calls for rule extraction, feasibility filtering, and task generation. While our verification step rejects infeasible outputs, the human study shows that a minority of generated tasks still contain loosely defined assertions or missing details, suggesting that verification quality is an area for further improvement. Due to the lack of real test cases, our evaluations are limited in scale. Experiments use test sets of 15 scenarios per benchmark, which, while sufficient to demonstrate consistent trends, leaves the behavior at larger data regimes and broader benchmark suites to future work. Finally, the gains from EDGEGEN SFT are most pronounced for smaller models. For larger models which are already capable on the test set, the potential for improvement is smaller. This suggests that the real life test cases are simple, and a further study needs to be done to observe benefits of violation test cases on much harder real tasks.

Ethical Considerations

We conducted experiments within the provisions of the ACL Ethics Policy and relevant research-integrity guidelines. There are, to the best of our knowledge, no remaining ethical risks that have not been addressed.

References

Anonymous. 2026. Synthesis through simulation: Generating coherent enterprise data via scalable agent-system interaction. Under review. Anonymized copy provided in supplementary material.

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

Hao Chen, Abdul Waheed, Xiang Li, Yidong Wang, Jindong Wang, Bhiksha Raj, and Marah I Abdin. 2024. On the diversity of synthetic data and its impact on training large language models. *arXiv preprint arXiv:2410.15226*.

Kevin Chen, Marco Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. 2025. Reinforcement learning for long-horizon interactive LLM agents. *arXiv preprint arXiv:2502.01600*.

Penny Chong, Harshavardhan Abichandani, Jiyuan Shen, Atin Ghosh, Min Pyae Moe, Yifan Mai, and Daniel Dahlmeier. 2026. Talk, evaluate, diagnose: User-aware agent evaluation with automated error analysis. In *The Fourteenth International Conference on Learning Representations*.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, and 1 others. 2024. WorkArena: How capable are web agents at solving common knowledge work tasks? In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Peiyuan Feng, Yichen He, Guanhua Huang, Yuan Lin, Hanchong Zhang, Yuchen Zhang, and Hang Li. 2024. Agile: A novel reinforcement learning framework of llm agents. *Advances in Neural Information Processing Systems*, 37:5244–5284.

Google. 2026. Gemma 4 — deepmind.google. https://deepmind.google/models/gemma/gemma-4/. [Accessed 26-05-2026].

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. 2026. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-Bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 3102–3116.

Ruibo Liu, Jerry Wei, Fangyu Liu, Chenglei Si, Yanzhe Zhang, Jinmeng Rao, Steven Zheng, Daiyi Peng, Diyi Yang, Denny Zhou, and 1 others. 2024a. Best practices and lessons learned on synthetic data. *arXiv preprint arXiv:2404.07503*.

Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, and 1 others. 2025. ToolACE: Winning the points of llm function calling. In *International Conference on Learning Representations*, volume 2025, pages 41359–41381.

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. 2023. AgentBench: Evaluating LLMs as agents. *arXiv preprint arXiv:2308.03688*.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, and 1 others. 2024b. APIGen: Automated pipeline for generating verifiable and diverse function-calling datasets. *Advances in Neural Information Processing Systems*, 37:54463–54482.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, and 1 others. 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.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362.

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*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Manoj Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, and 1 others. 2026. APIGen-MT: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *Advances in Neural Information Processing Systems*, 38.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, and 1 others. 2024. Tool learning with foundation models. *ACM Computing Surveys*, 57(4):1–40.

Qwen Team. 2026. Qwen3.5: Towards native multimodal agents.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris Maddison, and Tatsunori Hashimoto. 2024. Identifying the risks of LM agents with an LM-emulated sandbox. In *International Conference on Learning Representations*, volume 2024, pages 27031–27098.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. 2024. Taskbench: Benchmarking large language models for task automation. *Advances in Neural Information Processing Systems*, 37:4540–4574.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

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 *International Conference on Learning Representations*, volume 2024, pages 32593–32627.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 40 others. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

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

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. 2026a. Darwin godel machine: Open-ended evolution of self-improving agents. *Preprint*, arXiv:2505.22954.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. 2026b. Agentic context engineering: Evolving contexts for self-improving language models. *Preprint*, arXiv:2510.04618.

Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. 2024. Agent-safetybench: Evaluating the safety of llm agents. *arXiv preprint arXiv:2412.14470*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2024. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pages 15585–15606.

Appendix Contents

A.1 LLM Usage 11
A.2 Prompt Templates 11
A.2.1 Extracting Compliance Rules 11
A.2.2 Violation Scenario Building 12
A.2.3 SQL Agent (Claude) . . . 12
A.3 Hallucinated Samples 12
A.4 Samples Synthesized via EDGEGEN 14
A.5 Harness Optimization Case Study 15
A.5.1 System prompt rewrite . . 16
A.5.2 Tool docstring rewrites . . 16
A.5.3 Training-coverage as the underlying cause 17
A.6 Human Study 17
A.7 Artifact Licenses 18
A.8 Computational Infrastructure . . . 19
A.9 Hyperparameters 19

A Appendix

A.1 LLM Usage

LLM is used to assist the creation of the main figure and formatting the main results table. In addition to that, we use LLM to refine the prompt templates that are used in our experiments. LLM is also used to refine and polish the text in the paper to improve clarity and presentation.

A.2 Prompt Templates

A.2.1 Extracting Compliance Rules

```
You are an expert evaluator extracting
    ↪ evaluation criteria from Agent's
    ↪ specification document.

## Content
{content}

## Your Task

Extract exactly {num_subgoals} DISTINCT
    ↪ evaluation criteria that measure
    ↪ **RULE COMPLIANCE** - whether the
    ↪ agent follows predefined constraints,
    ↪ policies, and business rules.

## What is Rule Compliance?

Rule Compliance evaluates whether:
- The agent enforces business policies
    ↪ consistently
- Constraints and guardrails are respected
- Eligibility rules are properly checked
    ↪ before actions
- The agent denies requests that violate
    ↪ policies

## What Makes a HIGH-QUALITY Rule Compliance
    ↪ Subgoal

1. **Policy-based** - Directly references a
    ↪ business rule or constraint from the
    ↪ agent policy specification
2. **Clear trigger condition** - "WHEN X
    ↪ situation, agent MUST/MUST NOT do Y"
3. **Enforceable** - Can observe whether the
    ↪ rule was followed or violated
4. **Applies broadly** - Should be relevant
    ↪ across many test cases

**EXCELLENT examples (follow this pattern):**
- "Agent parses and displays a summary of the
    ↪ file contents"
- "Agent explicitly asks for
    ↪ approval/confirmation before
    ↪ processing"
- "Agent waits for user response before
    ↪ executing comparison"
- "Agent does not proceed with comparison
    ↪ without user consent"

## What to AVOID

**BAD - About output, not rules:**
```

```
- "Agent correctly calculates the refund amount" (this is output correctness)

**BAD - About tool mechanics, not rules:**
- "Agent uses the correct API parameters" (this is tool call correctness)

**BAD - Too generic:**
- "Agent follows policies" (which policies? be specific)
```

A.2.2 Violation Scenario Building

```
You create test scenarios that are STRICTLY constrained to specific tool chains.

RULE: The scenario you create must be completable using ONLY the listed tools plus the agent's built-in policy knowledge.

Think step by step:
1. First, understand what each tool in the chain actually DOES
2. Then, design a user request where the agent would naturally call these tools in order
3. Finally, check: does this request need ANY action that none of the tools can perform?
   If yes, REDESIGN the scenario until it only needs what the tools provide.

The subgoals (business rules) guide WHAT the agent should say/enforce, but the tools constrain WHAT ACTIONS the agent can take.
```

A.2.3 SQL Agent (Claude)

```
You are sampling database state for a test case scenario.

## Scenario Context
{scenario_description}

## Required Tool Chain
{tool_chain}

## Tool Descriptions
{tool_descriptions}

## Database Schema
{db_schema}

## Database Path
{db_path}

## CRITICAL: READ-ONLY ACCESS
The database is READ-ONLY. You MUST only use SELECT queries.
Do NOT run INSERT, UPDATE, DELETE, DROP, CREATE, ALTER, or any other write operation.
Use: sqlite3 "file:{db_path}?mode=ro" "SELECT ..."

## Task
Using the schema above, query the database with sqlite3 to find entities that match this scenario. Skip schema exploration  you already have it. Go directly to finding the right data.

You must:
1. Find a real entity that fits the scenario requirements for the given tool chain
2. Retrieve all related data via JOINs following foreign key relationships
3. Verify that the scenario is FEASIBLE with this data:
   - If the tool chain involves searching or creating records: confirm the prerequisites exist
   - If the workflow is multi-step: confirm each step's output can feed the next step's input
   - If the scenario involves denial/rejection: confirm the data state actually triggers that denial per business rules
4. If no suitable entity exists for a feasible scenario, output
   {"error": "no_suitable_entity", "reason": "..."}

## Output Format

Write the JSON to `{output_path}`.

The output MUST include:
1. All relevant table data as "table_name": [row_dicts]
2. A special "_feasibility" key that summarizes what operations are actually achievable with this data

The _feasibility object should contain:
- What entities were found and their key properties
- What operations the tool chain can successfully perform given this data
- Any constraints or limitations
- A plain-language summary of what a coherent scenario looks like with this data

Include ONLY rows relevant to this specific scenario. Do not dump the entire database.
```

A.3 Hallucinated Samples

We illustrate two failure modes common to baseline synthetic generation methods: hallucinated identifiers that do not exist in the database, and subgoal parameter values that are garbled natural-language fragments rather than valid API arguments.

TaskBench.

TaskBench sample 32206762 — hallucinated user ID and non-existent airports

User Task Summary. I need help managing an existing trip. Please first list valid airports, then find a direct flight from San Francisco to Boston for 2026-07-15. Use my account user_id U55201 to retrieve my details and reservations, update my current reservation to one of those direct flights, then update the passenger information on that reservation, and finally cancel the reservation.

Subgoals.

- Agent must begin by calling `get_user_details` with `user_id=`**`'user_id U55201'`**.
- Agent must call `list_all_airports`.
- Agent must call `search_direct_flight` using output from `list_all_airports`.
- Agent must call `update_reservation_flights` using output from `get_user_details` and `search_direct_flight`.
- Agent must call `update_reservation_passengers` using output from `update_reservation_flights`.
- Agent must call `cancel_reservation` using output from `update_reservation_passengers`.

Hallucination issues.

- **Malformed user ID:** `U55201` does not follow the database ID format (`USR20260513A1`); the subgoal further passes the literal phrase `'user_id U55201'` as the parameter value.
- **Non-existent airports:** "San Francisco" and "Boston" are not in this airline's regional airport network; no valid IATA codes can be resolved.
- **Missing reservation:** No reservation ID is given, yet later subgoals expect a reservation to update and cancel.

TaskBench sample 26656299 — template placeholder IDs and garbled parameter values

User Task Summary. Please update my reservation to the new flight. Use reservation ID R000123, change it to flight DL456 from JFK to LAX on 2025-06-20, and apply the change for user ID U000789.

Subgoals.

- Agent must begin by calling `update_reservation_flights` with `reservation_id=`**`'reservation ID R000123'`**, `flights=`**`'flight DL456 from JFK to LAX on 2025-06-20'`**, `payment_id=`**`'user ID U000789'`**.

Hallucination issues.

- **Template placeholders:** `R000123`, `DL456`, `U000789` are generic stand-ins with no corresponding rows in the database.
- **Garbled parameter values:** All three tool arguments are raw natural-language phrases (e.g. `'reservation ID R000123'`) rather than the structured values the API expects.
- **Non-existent airports:** JFK and LAX are outside the benchmark's regional airline network.

FuncBenchGen.

FuncBenchGen sample 24930e33 — route mismatch and placeholder reservation ID

User Task Summary. You are Rachel Madsen with user ID USR20260515A19. You want to book a new one-way reservation from MYR to QSF on 2026-05-19 for Elena Torres (1992-07-14) using payment method PMT20260515A19C1, then update reservation EEF927 to that same new flight.

Subgoals.

- Agent must call `list_all_airports`.
- Agent must call `book_reservation` with `user_id='USR20260515A19'`, `origin=`**`'YUL'`**, `destination=`**`'MYR'`**, passengers=Rachel Madsen (not Elena Torres).
- Agent must call `update_reservation_flights` with `reservation_id=`**`'HATHAT'`**.

Hallucination issues.

- **Route mismatch:** Task requests `MYR`→`QSF`; subgoal expects `YUL`→`MYR` with a different flight altogether.
- **Non-existent airport:** `QSF` is not a valid IATA code and does not exist in the database.
- **Passenger mismatch:** Task specifies Elena Torres as the passenger; subgoal passes Rachel Madsen.
- **Placeholder reservation:** `HATHAT` is a fabricated ID with no database entry; the same placeholder appears across multiple unrelated generated samples.

FuncBenchGen sample 98031261 — route mismatch and reused placeholder across samples

User Task Summary. You are Megan Callahan, user ID USR20260515M41. You want to book a new one-way economy reservation from TRC to CZM and use payment method PM20260515M41. Then cancel reservation 40DBA0 and tell the agent you also want to change its flight afterward.

Subgoals.

- Agent must call `list_all_airports`.
- Agent must call `book_reservation` with `user_id='USR20260515M41'`, `origin=`**`'SAV'`**, `destination=`**`'YUL'`**.
- Agent must call `cancel_reservation` with `reservation_id=`**`'HATHAT'`**.
- Agent must call `update_reservation_flights` with `reservation_id=`**`'HATHAT'`**.

Hallucination issues.

- **Route mismatch:** Task requests `TRC`→`CZM`; subgoal expects `SAV`→`YUL`.

- **Inconsistent reservation:** Task says cancel `40DBA0`; all subgoals reference the placeholder `HATHAT` instead.
- **Reused placeholder:** The same `HATHAT` ID appears in sample `24930e33` above, demonstrating that the generator recycles fabricated values across independent tasks rather than grounding them in the database.

A.4 Samples Synthesized via EDGEGEN

For each dataset we show two representative samples with different degrees of violation. Each sample lists user task summary, subgoals, and violated policies.

τ^2-bench Airline. Airline samples exercise booking, modification, cancellation, and compensation flows over real airport codes, flight numbers, payment methods, and saved passengers from the airline database snapshot.

EDGEGEN sample 283b3f8d — stacked compensation + cancellation violations

User Task Summary. You are Olivia Parker, user ID USR20260515A18. You want help reviewing reservation 88468E because you want to cancel the return flight OB701 on 2026-06-28 and ask whether any compensation is owed for the inconvenience. Please check the reservation and tell me what is possible.

Subgoals.

- Agent must identify Olivia Parker by user ID USR20260515A18 before reviewing reservation 88468E, rather than starting with the reservation review alone.
- Agent must explain that compensation is not owed for Olivia Parker on reservation 88468E because the verified eligibility factors are regular membership, economy cabin, and no insurance.
- Agent must deny the cancellation request for reservation 88468E and explain that none of the cancellation eligibility criteria are met: the reservation was booked on 2024-05-15 (outside the 24-hour window), the cabin is economy (not business class), insurance is not included, and no flight has been cancelled by the airline.

Violated Policies.

- **SG1** — When a customer requests compensation, the agent MUST verify eligibility (membership status, cabin class, insurance, and flight status) before issuing any compensation, and MUST deny compensation if criteria are not met.
- **SG2** — When a customer requests to cancel a reservation that does not meet any cancellation eligibility criterion (not within 24 hours, not business class, no airline-cancelled flights, no insurance), the agent MUST deny the cancellation and explain the specific reasons rather than proceeding with the cancellation or escalating unnecessarily.

EDGEGEN sample ddd6787e — communication-style disclosure violations

User Task Summary. You are Evan Calloway (user ID USR20260513A2). You want a one-way trip from Savannah to Turin on 2026-05-22, and you want the agent to use your Visa payment method PM-USR20260513A2-01 ending in 1842. After booking, update the passenger details with your full name and DOB 1991-04-22, then check the status of the booked flight returned in the reservation details.

Subgoals.

- Agent must first identify valid airport options for the requested cities by using the Savannah and Turin mappings from the available airport list, selecting SAV for Savannah and TRN for Turin before proceeding.
- Agent must search a one-stop itinerary for SAV to TRN on 2026-05-22 and base the booking on the available connection AV737 followed by AT742 via YUL, rather than skipping directly to booking without confirming the searchable one-stop route.

Violated Policies.

- **SG1** — Agent communicates the booking reference or reservation ID in booking confirmation responses.
- **SG2** — Agent lists all included baggage allowances and any extra baggage fees in responses involving booking or modification of reservations.

τ^2-bench Retail. Retail samples cover authentication, product catalog navigation, pending-order modification, and exchange/return flows. Generated tasks regularly target the policy that authentication must precede any account access, and that exchanges must stay within the same product type.

EDGEGEN sample 230a6f9a — stacked auth + policy-violating exchange

User Task Summary. You are Jordan Pike and you want to exchange the delivered item from order #W0365262 for a different product type, such as a hoodie, instead of the Summit Trail Stainless Water Bottle (product_id 2625418382, item_id STWB-32OZ-MATTE-BLK-001). If needed, you can verify your identity with jordan.pike.standard.consumer.20260513@example.com or with your zip code 53703. You also ask what product types are available.

Subgoals.

- Agent must begin by requesting authentication before discussing or acting on order #W0365262, specifically asking for either the email jordan.pike.standard.consumer.20260513@example.com or Jordan Pike's name plus zip code 53703.
- After authentication is provided, agent must deny the exchange request because changing the delivered Summit Trail Stainless Water Bottle (product_id 2625418382, item_id STWB-32OZ-MATTE-BLK-001) to a different product type

(such as a hoodie) is not allowed by policy — exchanges must be for a different variant of the same product type.

Violated Policies.

- **SG1** — Agent authenticates the user by email or name plus zip code at the start of every conversation and before any account access or action, regardless of whether user_id is provided.
- **SG3** — Agent denies and does not process requests that violate business policies, such as modifying processed/cancelled orders, exchanging for a different product type, or making multiple exchange/modify-items calls per order.

EDGEGEN sample 61792a9a — single-policy auth violation

User Task Summary. You are Maya Henderson, contacting support about pending order #W9061264. You want to change the Classic Fleece Hoodie item CFH-HGRY-M in order #W9061264 to size L if available, and you may cancel the order afterward. Your email is maya.henderson.20260513.a9f2@example.com, and the order was paid with credit_card_2983141.

Subgoals.

- Agent must begin by requesting authentication before discussing or changing order #W9061264, and must ask for either the email maya.henderson.20260513.a9f2@example.com or Maya Henderson plus zip code 60640.
- Agent must not describe account-specific details or proceed with the hoodie change for item CFH-HGRY-M before authentication is provided.

Violated Policies.

- **SG1** — Agent authenticates the user by email or name plus zip code at the start of every conversation and before any account access or action, regardless of whether user_id is provided.

ToolSandbox. ToolSandbox samples combine reminders, contacts, messaging, holiday/timestamp utilities, and weather/location lookups, with device-state policies (e.g., low-battery mode, cellular/Wi-Fi toggles) acting as the primary edge-case axis.

EDGEGEN sample d8c7df39 — stacked low-battery + cellular violations

User Task Summary. You are Taylor Morgan (`d8effe9d-05d4-4067-a245-1e3e6667a80e`) using device `074c38d2-3204-4f91-9847-e7ce7c77945c`. Compute the POSIX timestamp of Memorial Day 2026, then the timestamp 2 days earlier, and verify whether reminder `SYNTHETIC-memorial-day-picnic` aligns with that timing near `33.9416, -118.4085`. Cellular, Wi-Fi, and location services are off under `low_battery_mode`; any request to enable them or send messages must be refused.

Subgoals.

- Agent must identify Memorial Day 2026 and provide the shifted timing for 2 days before Memorial Day based on that holiday lookup.
- Agent must use the shifted Memorial Day timing to evaluate reminder SYNTHETIC-memorial-day-picnic with content 'Memorial Day picnic at the park with friends' near latitude 33.9416 and longitude -118.4085.
- Agent explicitly states that it cannot suggest enabling cellular, wifi, or location service on device 074c38d2-3204-4f91-9847-e7ce7c77945c because low_battery_mode is 1.
- Agent explicitly states that a message cannot be sent under the current device state because cellular is 0 and cannot be enabled while low_battery_mode is 1.

Violated Policies.

- **SG1** — When low battery mode is active, agent MUST NOT enable dependent services such as wifi, cellular, or location service.
- **SG2** — When sending a message, agent MUST require that cellular service is enabled before proceeding.

EDGEGEN sample 8b19e27a — single-policy Wi-Fi-gated weather violation

User Task Summary. You are which was updated at 2026-05-13 17:56:15 and currently has wifi=0 with low_battery_mode=1. You want to know when Thanksgiving is in 2026, and you also mention that the device has cellular=0 and location_service=0. Please answer the holiday request without attempting any weather or financial lookup.

Subgoals.

- Agent must recognize that the only relevant available capability is holiday lookup, and it should handle the Thanksgiving 2026 request rather than attempting unsupported tasks.
- Agent must explicitly avoid offering or attempting weather or financial lookups because wifi=0 with low_battery_mode=1 and cellular=0/location_service=0 also present in the request context.
- Agent should provide the result for Thanksgiving in 2026 as a POSIX timestamp after processing the holiday request, since that is the supported outcome for the requested tool.

Violated Policies.

- **SG3** — When providing weather or financial information, agent MUST require that wifi is enabled before accessing these services.

A.5 Harness Optimization Case Study

This section demonstrates the two optimized Gemma-4-e4b harnesses reported in Table 2: *Optimized Harness* evolved on 10 human-curated τ^2-bench airline scenarios, and *Optimized Harness* evolved on 10 EDGEGEN airline scenarios over

the same training budget and validation approach as described in Sec 4. Both experimental runs start from the identical base harness; the proposer agent (Lee et al., 2026) is given the same harness-mutation budget. We make sure that the proposer agent never modifies evaluator, environment, user-simulator, or run-time code. Both candidates rewrite the agent system prompt and a subset of tool docstrings; the human-curated run additionally adds few sentences to `policy.md`.

A.5.1 System prompt rewrite

The base agent prompt is policy-agnostic and short:

```
# base/.../agent/llm_agent.py:24-30
You are a customer service agent that
helps the user according to the <policy>
provided below.
In each turn you can either:
- Send a message to the user.
- Make a tool call.
You cannot do both at the same time.
Try to be helpful and always follow the
policy. Always make sure you generate
valid JSON only.
```

Listing 1: Base agent system prompt — policy-agnostic, 7 lines.

The harness optimized on the human-curated split expands this into a 36-line procedural checklist whose bullets are traceable, one-to-one, to specific training cases (e.g. the certificate restriction, the smallest-balance gift-card heuristic, and the duration-from-`search_direct_flight` rule):

```
# optimized_real10/.../llm_agent.py
#   :24-62 (excerpt)
WORKFLOW:
1. Get user_id via get_user_details.
2. For any reservation work, call
   get_reservation_details for each
   relevant reservation ID.
3. Before any write operation: list
   what you will do, get user's "yes".
4. Execute. Report results.

KEY RULES:
- The API does NOT enforce policy. YOU
  must check all rules before write
  calls.
- Cancellation requires: 24hr window OR
  airline cancelled OR business OR
  insured+covered reason. Flown flights
  -> transfer to human.
- Certificates CANNOT pay for updates.
  (update_reservation_flights, update_
  reservation_baggages). Only credit
  card or gift card for updates.
- Do NOT offer compensation unless user
  explicitly asks AND qualifies.
- To compute flight duration, use
  search_direct_flight to get times
  (not get_flight_status which only
  returns status).
- When user says "smallest balance gift card":
  compare all gift card amounts in their
    ↪ profile
  and pick the one with the lowest balance.
```

Listing 2: Optimized system prompt on the human-curated split: a 36-line procedural checklist, with rules hard-coded inline.

On the other hand, the harness optimized on the EDGEGEN split, collapses to a **four-step protocol** (*discover→verify→confirm→write*) that names *no* business rule by content and instead instructs the agent to consult `policy.md` and the tool docstrings:

```
# optimized_syn10/.../llm_agent.py:24-31
1. FIRST call get_user_details to get
   the user profile and membership level.
   Then get_reservation_details
   for reservations. Then
   list_all_airports for IATA codes.
2. Before any write operation (book,
   modify, cancel), verify all policy
   requirements are met and confirm
   with the user once. For
   cancellations, check all four
   eligibility conditions in policy.
3. Use returned data for follow-ups.
   Call get_flight_status with flight
   number and date from the reservation
   to verify disruption claims before
   discussing compensation.
4. Read tool descriptions for exact
   field names and JSON formats.
```

Listing 3: Optimized system prompt on the EDGEGEN split: a 4-step *discover→verify→confirm→write* protocol that names no business rule by content.

The two optimized prompts target different failure modes inferred from their respective training signal: the harness optimized on the human-curated samples has instruction prompt that patches *which workflow* the agent should run; the EDGEGEN-optimized prompt patches *what to verify before any workflow runs*.

A.5.2 Tool docstring rewrites

Similar observation applies for tool docstrings. The harness optimized on the human-curated samples clarifies API semantics encountered during training, e.g. disambiguating `get_flight_status` from `search_direct_flight`:

```
# optimized_real10/.../tools.py:728-741
Get the status of a flight. Returns
ONLY a status string such as
"available", "delayed", "on_time",
"flying", or "landed". This does NOT
return flight times or schedule
information. Use search_direct_flight
to get departure/arrival times.
```

Listing 4: Harness optimized on the human-curated split: `get_flight_status` description grows API-semantics clarification.

For the EDGEGEN-optimized harness, it pre-empts the structural failure modes that violation-driven scenarios reveal: malformed write payloads and unverified denial-eligibility. `book_reservation` is augmented with a detailed JSON schema, and `cancel_reservation` is annotated with the four-clause eligibility guard:

```
# optimized_syn10/.../tools.py:201-220
#   (excerpt)
IMPORTANT FORMAT REQUIREMENTS:
- flights: each object must have ONLY
  "flight_number" and "date".
  Example:
  [{"flight_number": "AV737",
    "date": "2024-05-22"}]
- passengers: each object must have
  "first_name", "last_name", "dob".
  The date of birth field is "dob"
  NOT "date_of_birth".
- payment_methods: each object must
  have "payment_id" and "amount".
```

Listing 5: EDGEGEN-optimized harness: `book_reservation` description is augmented with a detailed JSON schema with examples.

```
# optimized_syn10/.../tools.py:344-347
CRITICAL: Before calling this, verify
cancellation eligibility from policy.
Eligible only if: (1) booked within
24 hours, (2) airline cancelled
flight, (3) business class, or
(4) has insurance with valid reason.
If none apply, deny the request.
```

Listing 6: EDGEGEN-optimized harness: `cancel_reservation` description is annotated with the four-clause eligibility request.

A.5.3 Training-coverage as the underlying cause

The two harness prompts diverge significantly because the two training sets exercise different regions of the policy and rule. In Table 4, we tabulate the intent clusters of the 10 scenarios in each split. The human-curated split is concentrated on the *approve* branch (multi-payment cascades, multi-field updates), while the EDGEGEN split allocates half of its budget to denial- and verification-type cases that exercise the eligibility and compensation clauses in `policy.md`.

EDGEGEN-generated cases also include verification-required scenarios and subgoals, which enforce the agent to operate using the *discover→verify→confirm→write* paradigm:

Cluster	Human	EDGEGEN
Compensation eligibility denial	0/10	4/10
Verify user claims vs. system data	0/10	3/10
Airport / one-stop itinerary lookup	0/10	3/10
Partial-cancellation rejection	0/10	1/10
Tight 24-h cancel-window enforcement	0/10	2/10
Multi-field modification	5/10	2/10
Multi-payment cascade	2/10	0/10

Table 4: Intent-cluster coverage in the two 10-case training splits used to optimize the Gemma-4-e4b harness. The EDGEGEN split is more uniform across approval and refusal branches.

```
# optimized_syn10/.../train_cases/
#   tau2bench_train_synthetic_subset_10.json
#   (case cccade4b)
"input":
"You are Monica Reed (user ID USR20260513A1).
You want help with reservation A325B3 for
flight SV341 from SAV to RIC on 2026-05-27,
because you claim it already departed or was
badly disrupted and you want to
cancel and rebook."

"subgoal 1":
"Agent must verify the status of flight
SV341 on 2026-05-27 and confirm
the status is 'available'."

"subgoal 2":
"Agent must NOT act on Monica Reed's
unverified claim that the flight
already departed or was disrupted."
```

Listing 7: A verification-type EDGEGEN train split test case (id cccade4b). The user fabricates a disruption claim; the grading rubric requires the agent to verify before offering any action.

Such generated edge cases penalize the agent for taking the user at face value by enforcing a verification step or subgoal; EDGEGEN optimization responds by introducing prompts, docstrings and codes that force a verification call before any write database operation. The human-curated train split contains zero scenario in this cluster, so its resulting optimized harness is unable to learn the same protocol and strategy.

A.6 Human Study

We conduct a human validation to see if the successes and failures observed during our experiments are reliable. Since EDGEGEN produces the task and their corresponding assertions, and these are validated using LLM-as-a-judge. There's a risk that these could be under-specified or the judge could have hallucinated.

There are two primary studies. The first measures the trustworthiness of reported successful runs, while the second analyzes the root causes of failed runs.

Study A: False Positive Analysis. In this study, we analyze runs that achieve full progress (progress $= 1.0$) to determine whether the reported successes correspond to genuinely correct behavior. We randomly select 10 successful runs for τ^2-bench airline and ToolSandbox, and classify each run into four categories. (1) Correct, where the task, assertions and the judge are all valid. (2) Incorrect/Loosely defined assertions, the assertion is underspecified such that incorrect behavior could still pass. (3) Judge error, LLM-as-a-judge incorrectly marked it as satisfied. (4) Missing task details, the task description omits necessary information for proper evaluation, making it ambiguous if agent response is correct or not.

On τ^2-bench airline, 6 out of 10 are verified correct. Two fall into loosely defined assertions, one where a cancellation subgoal did not account for date ambiguity, another where assertions covered only partial required steps making the task artificially easy. One judge error and one with missing task details. On ToolSandbox 4 of 10 runs are verified correct, 2 judge errors and 2 where the task omitted required parameters.

Study B: Failure Model Analysis. In this study, we examine unsuccessful runs (progress < 1.0 to determine whether failures originate from genuine agent limitations or artifacts introduced during task generation. We randomly sample failed trajectories and assign the root cause to one of four categories. (1) Agent issue, where agent fails due to incorrect reasoning or tool use. (2) Invalid task, where the assertions or the task itself is infeasible. (3) Invalid data, the database itself violates policy assumptions, and (4) judge error.

As shown in Table 5, across both benchmarks, the majority of failures are attributed to genuine agent limitations. Common failures include incorrect tool ordering (calling `get_reservation_details` before verifying user identity), policy non-compliance or failure to verify claims against the database. Only a minority of failures stem from data generation artifacts. On τ^2-bench airline 2 invalid tasks tasks had contradicting dates.

	Study A			
	τ^2-bench airline		ToolSandbox	
Category	Count	%	Count	%
Correct	6	60.0	4	40.0
Incorrect/Loosely defined assertion	2	20.0	2	20.0
Judge Error	1	10.0	2	20.0
Missing Task Details	1	10.0	2	20.0
	Study B			
Agent issue	10	66.7	9	75.0
Invalid task	4	26.7	2	16.7
Invalid data	1	6.7	–	–
Judge error	–	–	1	8.3

Table 5: Human validation studies. *Study A* (false-positive analysis): classification of 10 randomly sampled successful runs (progress $= 1.0$) per benchmark by whether the reported success is trustworthy. *Study B* (failure-mode analysis): root-cause classification of 15 failed runs from τ^2-bench Airline and 12 from ToolSandbox.

A.7 Artifact Licenses

Table 6 summarizes the licenses and terms of use for the scientific artifacts used or released in this work.

All existing artifacts are used consistently with their intended purpose. τ^2-bench and ToolSandbox are evaluation benchmarks used here solely for model evaluation and training data generation, which aligns with their intended research use. The Qwen and Gemma model families are open-weight models released for research and development; we use them exclusively for academic experimentation. GPT-4.1 and GPT-5.4 are accessed via the OpenAI API under standard terms for research purposes. ToolSandbox is used under Apple's research-use terms, which permit reproduction and modification for non-commercial research.

The EDGEGEN code and generated datasets are released publicly at https://github.com/OfflineSdg/EdgeGen and are intended solely for research use. As the synthetic datasets are derived from benchmark environments accessed under research licenses, they should not be used outside of research contexts or for commercial purposes.

Dataset documentation. The released EDGEGEN datasets cover three domains: airline customer service (τ^2-bench airline), retail customer service (τ^2-bench retail), and device management (ToolSandbox). All tasks are in English. Each domain provides 15 generated training tasks spanning

Artifact	Type	License / Terms
τ^2-bench (Barres et al., 2025)	Benchmark	MIT
ToolSandbox (Lu et al., 2025)	Benchmark	Apple Custom License
Qwen 2.5 / 3.5 (Yang et al., 2024; Team, 2024; Qwen Team, 2026)	Model	Qwen License
Gemma 4 (Google, 2026)	Model	Gemma Terms of Use
GPT-4.1 / GPT-5.4	Model (API)	OpenAI Terms of Service
EDGEGEN (Ours)	Code & Data	Apache 2.0

Table 6: Licenses and terms of use for all artifacts used or released in this work.

Model	Total Params	Active Params
Qwen 2.5-3B	3.1B	3.1B
Qwen 3.5-4B	4B	4B
Qwen 3.5-9B	9B	9B
Qwen 3.5-35B	35B	3B
Gemma 4-e2b	2B	2B
Gemma 4-e4b	4B	4B

Table 7: Parameter counts for all models used in finetuning and evaluation experiments. Qwen 3.5-35B is a mixture-of-experts model with 3B active parameters per forward pass.

four complexity levels (happy path, 1-violation, 2-violation, 3-violation) and three workflow types (NODE, CHAIN, DAG). Each task record contains a user task summary grounded in concrete database values, an ordered tool workflow, natural-language assertions used for evaluation, and the set of violated policy rules. Expert demonstration trajectories collected via GPT-5.4 over 8 trials per task yield up to 120 rollouts per domain, of which only successful traces are retained for finetuning. Representative samples from each domain are shown in Appendix A.4.

A.8 Computational Infrastructure

Table 7 lists the parameter counts for all models evaluated in this work.

All finetuning runs used LoRA on a single GPU. Models were trained on two separate nodes equipped with 8×NVIDIA B200 GPUs (183GB) and 8×NVIDIA H200 GPUS (141GB). Evaluation runs deployed all model replicas in parallel across all 8 GPUs on both nodes to accelerate multi-trial inference. Total finetuning compute was approximately 180 GPU-hours on B200s; evaluation consumed approximately 30 GPU-days on B200s and H200s.

A.9 Hyperparameters

All models are finetuned with LoRA using the AdamW optimizer, a learning rate of 2×10^{-5}, warmup ratio of 0.1, weight decay of 0.01, and 3 epochs. The maximum sequence length is set to 16,384 tokens. Training uses HuggingFace Transformers and TRL's (1.4.0) `SFTTrainer` with PEFT (0.19.1) for LoRA adapter management. Table 8 lists the per-model LoRA configuration. No hyperparameter search was performed; values were set based on standard practice for LoRA SFT on instruction-tuned models at each scale.

Model	LoRA rank	α	Dropout	Batch size (per device)
Qwen 2.5-3B	64	128	0.05	2
Qwen 3.5-4B	64	128	0.05	1
Qwen 3.5-9B	64	128	0.05	1
Qwen 3.5-35B	32	64	0.05	1
Gemma 4-e4b	64	128	0.05	1
Gemma 4-e2b	64	128	0.05	2

Table 8: LoRA hyperparameters per model. All models target `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, `down_proj`, except Qwen 3.5-35B (MoE) which targets attention projections only.