User-Oriented Multi-Turn Dialogue Generation with Tool Use at scale

Anonymous ACL submission

Abstract

The recent paradigm shift toward large reasoning models (LRMs) as autonomous agents has intensified the demand for sophisticated, multi-turn tool-use capabilities. Yet, existing datasets and data-generation approaches are limited by static, predefined toolsets that cannot scale to the complexity of open-ended human-agent collaboration. To address this, we initially developed a framework for automated task-oriented multi-turn dialogue generation at scale, utilizing an LRM-based simulator to dynamically generate high-value, domain-specific tools to solve specified tasks.

However, we observe that a purely task-oriented design often results in "solely task-solving" trajectories, where the agent completes the objective with minimal interaction, failing to generate the high turn-count conversations seen in realistic scenarios. To bridge this gap, we shift toward a user-oriented simulation paradigm. By decoupling task generation from a dedicated user simulator that mimics human behavioral rules—such as incremental request-making and turn-by-turn feedback—we facilitate more authentic, extended multi-turn dialogues that reflect the iterative nature of real-world problem solving. Our generation pipeline operates as a versatile, plug-and-play module capable of initiating generation from any state, ensuring high scalability in producing extended tool-use data. Furthermore, by facilitating multiple task completions within a single trajectory, it yields a high-density dataset that reflects the multifaceted demands of real-world human-agent interaction.

1 Introduction

The evolution of large language models (LLMs) has reached a critical inflection point, transitioning from text-generative systems into large reasoning models (LRMs) acting as autonomous agents (Guo et al., 2025; Yang et al., 2025a; Yehudai et al., 2025). This change is driven by advances in both *reasoning* and *tool use*, grounded in a core set of agentic capabilities: the high-level decision-making and planning required to decompose complex tasks; the technical precision of tool choice and argument generation; the analytical rigor of result analysis and error handling; and the foundational memory and environment awareness needed to maintain context in dynamic settings (Wang et al., 2023; Liu et al., 2023; Mialon et al., 2023a; Wu et al., 2024; Wang et al., 2024; Xi et al., 2025a). While tool use grounds reasoning in the real world, the ultimate objective of LRMs is to orchestrate these interdependent capabilities to support extended, multi-turn interactions that reflect the dynamics of real-world human–agent collaboration.

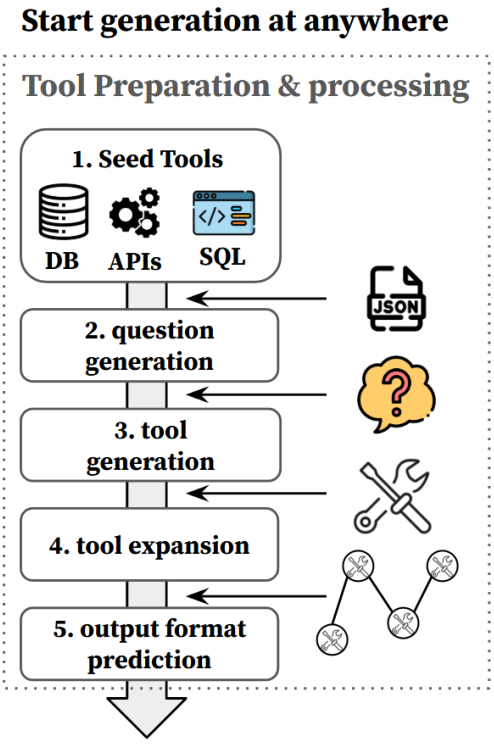

Figure 1: Plug-and-Play Tool Preparation Module. A modular pipeline for dynamic tool synthesis and preprocessing, designed to initiate multi-turn data generation from any arbitrary state.

Despite the growing capabilities of LRMs, progress remains constrained by the lack of high-quality and diverse training data. Most existing datasets rely on static, predefined toolsets, which inadequately capture the open-ended and evolving nature of real-world human–agent collaboration (Team et al., 2025a; Zhang et al., 2025; Prabhakar et al., 2025). Agents trained under such fixed schemas often struggle to generalize beyond seen domains or to reason over unfamiliar tool compositions. Moreover, many data-generation pipelines implicitly favor single-shot trajectories[1], in which a user poses a complex request and the agent responds with an optimal tool-use sequence in a single task. While efficient, these interactions fail to reflect the iterative, incremental, and often noisy nature of real-world human–agent collaboration.

To overcome these limitations, we first developed an automated framework for large-scale task-oriented dialogue generation. Leveraging an LRM-based simulator, the framework dynamically synthesizes domain-specific tools and database schemas (e.g., SQL-style read/write operations), along with corresponding tasks and evaluation rubrics. While this successfully scaled the volume of data, we observed an efficiency trap: the simulator, acting as a perfect task-solver, tended to complete objectives with the minimum number of turns. These solely task-solving trajectories lacked the back-and-forth dialogue—clarifications, incremental requests, and feedback loops—that define realistic human interaction.

To address this efficiency-driven bias, we propose a user-oriented simulation paradigm. Our approach decouples the objective (the "Task") from the interaction (the "User"). By employing a dedicated user simulator governed by human behavioral rules—such as asking for only one subtask at a time and providing turn-by-turn feedback—we force the agent to navigate extended, multi-turn dialogues.

Our generation pipeline consists of three key components: (1) Dynamic Tool & Task Synthesis: Instead of relying on fixed APIs, our LRM-based generator creates unique, rubric-backed tasks grounded in synthesized database schemas, ensuring the agent learns to reason over diverse structures. (2) Plug-and-Play Scalability: The generation pipeline is modular. It can initiate a simulation from any state—whether starting from a blank slate or injecting a tool-use requirement into an ongoing conversation—making it highly versatile for data augmentation. (3) High-Density Trajectories: By allowing multiple task completions within a single conversation thread, we produce a "high-density" dataset. This reflects the multifaceted nature of real-world use cases, where a user might update a record, query a trend, and request a summary all within a single session.

Empirical results on agentic benchmarks, including BFCL (Patil et al.) and $\tau 2$ (Barres et al., 2025), demonstrate that models trained on our data achieve consistently stronger multi-turn performance and more reliable tool usage, particularly in long-horizon and stateful domains. Moreover, consistency analysis under repeated executions shows that our models sustain correct tool-use behavior across multiple trials, rather than relying on isolated successes. Our findings highlight the importance of user-oriented interaction modeling and execution-grounded supervision for training robust and realistic agentic reasoning models.

2 Related Works

Reasoning Models and Tool-use Benchmarks. Recent advancements in LRMs have catalyzed the development of benchmarks designed to evaluate autonomous agents in tool-mediated environments (Guo et al., 2025; Yang et al., 2025a; Team et al., 2025a,b; Zeng et al., 2025). Early benchmarks primarily focused on single-turn tool invocation or static API selection within constrained domains (Mialon et al., 2023b; Qin et al., 2023; Lee et al., 2025). However, as the field shifted toward more complex problem solving, datasets like StableToolBench (Guo et al., 2024), BFCL (Patil et al., 2024), and τ benchmarks (Yao et al., 2024; Barres et al., 2025) emerged to test the model's ability to navigate vast API landscapes (Liu et al., 2025; Xu et al., 2025; Xi et al., 2025b).

Evolution of Tool-use Agents. Beyond benchmarking, the paradigm for tool-use agents has evolved from simple function-calling to sophisticated, autonomous orchestration. While early frameworks enabled LLMs to parse queries and interpret results, they often relied on fixed toolsets, which inherently limited their adaptability to open-ended tasks (Schick et al., 2023; Hao et al., 2023). To address this, recent research has explored the dynamic creation of tools, such as generating reusable tools on the fly (Cai et al., 2023) or leveraging exist-

[1]We define the trajectory as a sequence of API Calls, which are related (or correlated) with tool arguments for solving tasks

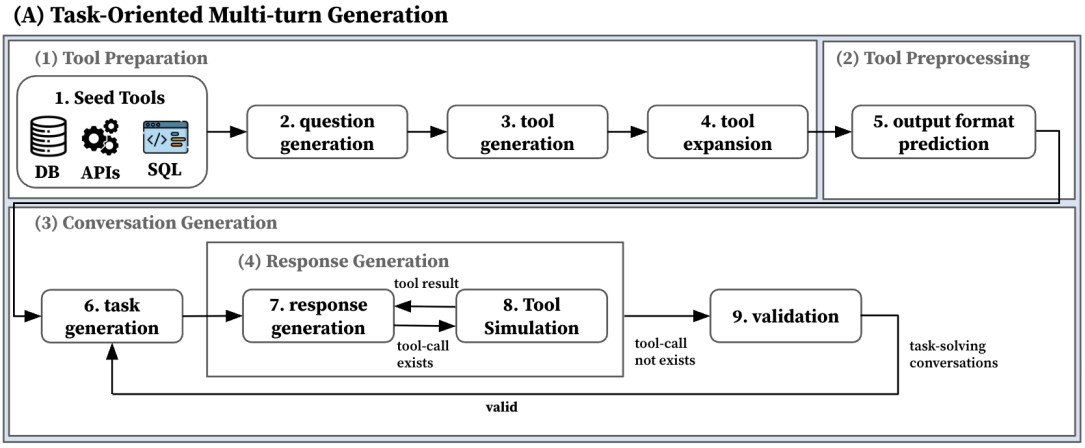

Figure 2: Task-Oriented Multi-Turn Conversation Generation Pipeline. An automated framework that generates tool-use trajectories focused on efficient task completion through direct simulator-based responses.

ing code repositories through ToolMaker (Wölflein et al., 2025). Furthermore, specialized training strategies have been proposed to enhance agentic capabilities, including critique-informed planning (Chen et al., 2025), fine-tuning on selective reasoning steps (Yang et al., 2025b), and decoupling reasoning from format following (e.g., Agent-FLAN) (Chen et al., 2024). Despite these advances, most existing approaches still struggle to maintain long-term coherence in multi-turn interactions, a gap that our user-oriented simulation framework aims to bridge.

Synthetic Dialogue Generation for Agents. The scarcity of training data still requires high-quality synthetic data generation. Despite these efforts, existing data generation approaches rely on fixed, predefined trajectories and toolsets (e.g., API graphs) (Mitra et al., 2024; Sengupta et al., 2024; Arcadinho et al., 2024; Tang et al., 2025) and rigid schemas that fail to capture the stochastic and iterative nature of real-world dialogues, such as clarifying ambiguous user intents or handling incremental feedback (Team et al., 2025a; Prabhakar et al., 2025; Zhang et al., 2025). Consequently, there remains a significant gap in evaluating how reasoning models maintain coherence (Barres et al., 2025) and adapt their tool-calling strategies over extended (Zhang et al., 2024), multi-turn interactions—a limitation that underscores the need for a more dynamic, user-oriented simulation paradigm.

In our work, we decouple the generation process into independent stages, each with an individual component to be replaced easily with just modifying input-output format. By architecting our pipeline as a versatile, plug-and-play module, we overcome the rigidity of previous approaches and enable the generation of high-density trajectories from any arbitrary state. This allows for the synthesis of authentic, extended dialogues that incorporate incremental request-making and iterative feedback loops. Consequently, our framework not only scales the production of domain-specific tools and database schemas dynamically but also ensures the generation of verifiable, multi-turn interactions that reflect the multifaceted and often noisy nature of real-world human-agent collaboration.

3 Task-Oriented Multi-turn Generation

To address the scarcity of high-quality agentic datasets, we developed a scalable, end-to-end pipeline designed to generate complex, multi-turn tool-use data. While existing datasets like Nemotron (Nathawani et al., 2025) provide a foundation with approximately 19K unique tools, they fall short of effective agentic training. Our pipeline automates the entire lifecycle of data generation—from tool creation to task validation. Designed as a plug-and-play module, the proposed framework allows for easy swapping of individual components by simply modifying input-output formats, ensuring diversity across domains and complexity in tool-interaction patterns. An overview of the pipeline is illustrated in Figure 2.

3.1 Tool preparation

The tool preparation stage aims to construct a diverse and realistic toolset from a minimal initial seed tool. We begin by generating realistic user questions inspired by the existing seed tools, such as databases and APIs, to ensure that the synthesized tools are grounded in practical use cases. Con-

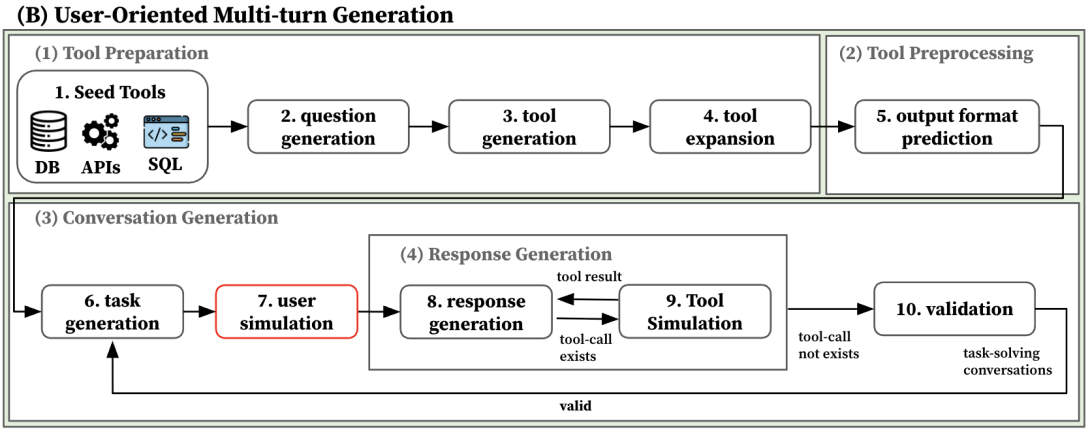

Figure 3: User-Oriented Multi-Turn Conversation Generation Pipeline. A framework that decouples tasks from interaction by employing a dedicated user simulator to mimic incremental human feedback and request-making.

ditioned on these questions, the model generates detailed tool specifications, including tool names, natural language descriptions, and required parameters, such that each tool can programmatically solve the intended task. To further expand domain coverage and interaction diversity, the framework analyzes the initial toolset to identify functional gaps and proposes up to ten complementary tools, resulting in a richer and more expressive toolset. We detailed the prompt in Appendix A.

3.2 Tool preprocessing

Before conversation simulation, we perform tool preprocessing to ensure structural and semantic consistency across the generated toolset. Specifically, the model is instructed to predict a JSON schema for the return value of each tool, making tool outputs explicit and machine-verifiable. Schema definitions are generated in a multi-turn conversational manner, allowing the model to reason over previously defined tools and maintain input-output consistency. As a result, shared entities such as `user_id` or `timestamps` preserve consistent data types and semantics across different tools and interaction turns.

3.3 Conversation Generation

Task Generation. Given the preprocessed toolset, we generate multi-turn conversations by constructing structured, rubric-based tasks. Following Kimi-K2 (Team et al., 2025a), each task is categorized by difficulty level (easy, medium, or hard) and is accompanied by a detailed rubric that specifies success criteria, expected tool-use patterns (with placeholders for dynamic arguments), and intermediate evaluation checkpoints. These components enable objective, step-level verification of agent behavior while encouraging complex reasoning and multi-step tool interactions.

Response Generation. In this stage, the generation model (here we use GPT-OSS-120b) produces responses that are validated for correctness and quality. Since all tools are synthetic, an LRM-based simulator is employed to generate tool execution results conditioned on the provided arguments and the evolving conversation context. To ensure temporal realism, the simulator maintains a randomized reference time while prioritizing user-specified temporal information when present.

Validation. Finally, a dedicated validation module compares the agent's responses against the predefined rubrics, filtering trajectories based on semantic correctness and required tool invocations. Only successful, high-density interaction trajectories are retained in the final dataset. Despite the effectiveness of this task-oriented pipeline, we observe that it produces trajectories focused solely on efficient task completion with minimal interaction. This limitation motivates the transition toward a user-oriented simulation paradigm, which we introduce in the following section.

4 User-Oriented Multi-turn Generation

Although the task-oriented pipeline effectively scales data volume, we observe that it frequently falls into an *efficiency trap*, where a highly capable simulator completes complex objectives in a single turn with minimal interaction (see statistics in Table 1). Such behavior produces trajectories that are optimized for task completion, failing to capture the incremental, exploratory, and iterative nature of

Source data	Step	Turn	Task	Samples
Task-oriented				
Nemotron	3.95 (89)	12.84 (178)	1.63 (18)	161,608
User-oriented				
Nemotron	3.45 (121)	21.79 (596)	2.48 (20)	177,375
Tau2	3.05 (63)	36.16 (780)	3.02 (20)	4,138
User-oriented + Tool-Execution				
Tau2	2.43 (23)	17.15 (294)	1.6 (10)	2,174
SQL	3.74 (322)	30.85 (680)	1.86 (11)	16,618

Table 1: Statistics of Generated Datasets. Comparison of conversation density (steps and turns) across task-oriented and user-oriented paradigms; values in parentheses represent the maximum observed counts. We define each column in Appendix B

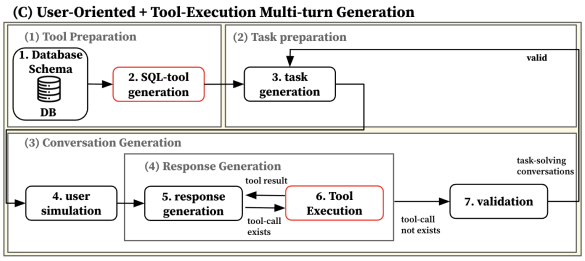

Figure 4: User-oriented Tool-Execution Multi-turn Conversation Generation Pipeline. This pipeline integrates a SQL-tool generation module grounded in real-world database schemas with a dedicated user simulator to produce verifiable, high-fidelity multi-turn dialogues.

realistic human–agent collaboration. In addition, many simulator-based approaches depend on synthetic tool outputs, which limits the faithfulness and verifiability of the resulting interactions.

To better reflect realistic usage patterns, we introduce a user-oriented multi-turn generation paradigm that explicitly models user behavior and interaction dynamics (see Figure 3). Building on the same tool-use abstraction introduced in earlier sections, this paradigm further grounds tool interactions in executable environments, enabling multi-turn trajectories whose intermediate states and outcomes are consistently maintained across turns rather than being implicitly assumed from a single-shot request.

Descriptive Task Generation. To support user-oriented interaction, we modify the generation pipeline to produce descriptive tasks instead of direct user questions. Rather than emitting a fully specified natural-language query, the model first generates a declarative statement that describes the user's ultimate objective in a self-contained manner. These descriptive tasks serve as latent goals that guide the user simulator, which must then realize them incrementally through multi-turn interaction.

For settings that involve structured tools, the task generator is additionally conditioned on concrete environment information, such as database schemas and limited data views. This grounding ensures that generated tasks remain feasible with respect to actual table structures and supported operations, including reading, updating, and combining records. To avoid introducing unsupported assumptions—particularly when only partial information is available—the generator is encouraged to phrase uncertain facts as information that must be retrieved or confirmed through subsequent tool use, rather than treating them as known in advance.

For longer trajectories, additional descriptive tasks are introduced either as natural extensions of previous tool outcomes or as independent but contextually coherent objectives within the same domain. Throughout this process, task complexity is explicitly controlled (easy, medium, or hard), ensuring that the resulting dialogues remain appropriately challenging while remaining consistent with earlier task-oriented settings.

User Simulation Interaction Loop. At the core of the proposed paradigm is a dedicated user simulator governed by simple yet expressive behavioral rules. Given a descriptive task, the simulator identifies the required sub-tasks but deliberately issues requests in a piecemeal fashion, typically asking for only one or two subtasks per turn. This design encourages the assistant to engage in intermediate reasoning, clarification, and verification, rather than converging immediately on a final answer.

Unlike simulated pipelines, the simulator conditions its behavior on tool outputs that are produced through actual execution. In practice, tools correspond to concrete operations such as parameterized database queries that are executed against a controlled environment, and their results are returned verbatim to the dialogue. The simulator maintains contextual awareness by reviewing the assistant's prior responses and tool outcomes to assess which components of the overall goal have been satisfied. Based on this assessment, it provides turn-by-turn feedback, requests clarifications, or introduces follow-up questions until the objective is fully achieved. A conversation is considered complete only when the simulator explicitly signals task completion by setting `is_task_complete` to true.

Model	BFCL					τ2		
	Multi-turn	Live	Non-Live	Hall. (Rel)	Hall. (Irrel)	Airline	Retail	Telecom
Proprietary Models								
Claude Sonnet 4.5 (fc)†	60.9	81.1	88.6	68.8	86.3	70.0	86.2	98.0
GPT-5.1-mini†	27.4	58.6	70.2	68.8	91.8	-	-	-
GPT-5.1†	36.1	59.0	73.0	68.8	91.4	77.9	-	95.6
Gemini-2.5-Pro†	29.3	63.8	85.6	43.8	91.5	-	-	-
Gemini-3.0-Pro†	-	-	-	-	-	73.0	85.3	85.4
Open-Sourced Models								
xLAM-2-3b-fc-r†	56.0	58.7	82.9	94.4	57.9	32.0	44.4	-
xLAM-2-8b-fc-r†	69.3	66.7	84.4	83.3	64.1	35.2	58.2	-
xLAM-2-32b-fc-r†	66.4	73.8	89.5	83.3	76.3	45.0	64.3	-
Qwen3-4B-Thinking-2507	48.1	82.9	86.3	100.0	78.9	46.0	56.1	21.1
Qwen3-30B-A3B-Thinking-2507	53.8	84.1	89.6	100.0	80.6	56.0	54.4	22.8
GPT-OSS-120b	51.3	72.6	37.5	75.0	85.5	56.4	75.3	59.5
Baselines								
Qwen3-4B-Thinking-2507 + APIGEN	50.9	83.1	87.5	83.3	82.3	50.0	58.8	30.7
Qwen3-30B-A3B-Thinking-2507 + APIGEN	53.8	83.5	90.1	83.3	82.3	56.0	60.5	33.3
Qwen3-4B-Thinking-2507 + NEMOTRON	52.1	85.8	88.2	75.0	85.5	44.0	50.9	26.3
Qwen3-30B-A3B-Thinking-2507 + NEMOTRON	46.0	50.9	88.7	88.9	76.3	54.0	48.2	28.1
OURS								
Qwen3-4B-Thinking-2507 + OURS	52.7	84.9	89.6	83.3	80.6	52.0	57.0	36.8
Qwen3-30B-A3B-Thinking-2507 + OURS	55.5	86.5	90.1	88.9	83.8	56.0	57.8	42.1

Table 2: Agentic benchmark results across proprietary and open-source models. † refers to the reported scores. The table compares performance on the Berkeley Function Calling Leaderboard (BFCL) (Patil et al.) and τ2 (Barres et al., 2025) benchmarks, highlighting the improvements gained from our data generation pipeline compared to baselines like APIGEN (Prabhakar et al., 2025) and NEMOTRON (Nathawani et al., 2025).

High-Density Multi-turn Trajectories. By allowing multiple descriptive tasks to be addressed within a single conversational thread, the proposed pipeline naturally produces high-density multi-turn trajectories. This setting mirrors realistic usage scenarios in which users perform a sequence of related actions—such as querying information, updating records, and requesting summaries—within a single session. To maintain coherence, state changes introduced by tool use persist across turns within the same trajectory, while remaining isolated across different generation instances.

Moreover, the modular and plug-and-play design of the pipeline enables generation to begin from arbitrary intermediate states, significantly improving scalability and diversity for extended tool-use data. Overall, the user-oriented paradigm complements the task-oriented pipeline by emphasizing interaction richness, temporal continuity, and verifiable tool use, which together are essential for training robust agentic reasoning models.

From Simulated Tools to Executable SQL-driven Agents. To overcome the scalability limits of static toolsets and the hallucination risks inherent in model-based simulations, we introduce a framework that synthesizes executable tool interfaces grounded in real-world relational databases (see Figure 4). By leveraging diverse schemata from open-source datasets like Spider (Yu et al., 2018), our pipeline automatically generates domain-specific functions mapped to complex SQL queries. We visualized the domains and examples used in our generated data in Appendix C and D. This approach allows the agent to interact with a functional database engine in real-time during the dialogue generation process, ensuring that the tool outputs used for training are computationally verified and factually accurate. Consequently, this SQL-backed synthesis enables the production of high-fidelity, multi-turn trajectories at scale, transforming the data generation pipeline from a closed-loop simulation into a verifiable, agentic execution environment.

5 Experiments

5.1 Experimental Setups

Training & Inference. We perform full fine-tuning of reasoning models. To balance the training data provided in Table 1, we downsampled the synthetic data generated from the Nemotron dataset due to the lower performance trend. All models are trained for five epochs, and the checkpoint corresponding to the best validation performance is selected for final evaluation. For inference, we serve the fine-tuned models using vLLM (Kwon et al., 2023), enabling efficient long-context decoding and

Generation Pipeline	BFCL	Tau2
	Multi-turn	Telecom
Task-oriented		
Qwen3-4B-Thinking-2507	50.9	24.5
Qwen3-30B-Thinking-2507	53.8	26.3
User-oriented		
Qwen3-4B-Thinking-2507	51.8	30.7
Qwen3-30B-Thinking-2507	54.5	34.2
User-oriented + Tool Execution		
Qwen3-4B-Thinking-2507	52.7	35.1
Qwen3-30B-Thinking-2507	54.9	40.4

Table 3: Ablation study of the generation pipeline.

high-throughput evaluation. Our experiments focus on two Qwen-family reasoning models (Yang et al., 2025a) with varying scales and generation model GPT-OSS-120b (Agarwal et al., 2025). This setup allows us to analyze the impact of model capacity on multi-turn, tool-augmented agent behavior. We detailed the rest of the descriptions in Appendix E.

Evaluation. To evaluate an agent's robustness to noisy, incremental user requests and its ability to sustain coherent multi-turn tool interactions, we adopt two complementary agentic benchmarks: $\tau 2$ (Barres et al., 2025) and the Berkeley Function Calling Leaderboard (BFCL) (Patil et al.). We detailed the evaluation sets in Appendix F.

Source Data and Statistics. As summarized in Table 1, we construct a high-density tool-use dataset by leveraging seed tools from the NEMOTRON (Nathawani et al., 2025) post-training dataset and structured tasks from the $\tau 2$ database benchmark. These seeds are expanded via our pipeline into diverse, domain-specific toolsets and executable database schemas. Unlike task-oriented datasets, our generated trajectories frequently contain multiple task completions within a multi-turn conversation. This design reflects realistic user sessions and enables more faithful training and evaluation of long-horizon agentic behavior.

5.2 Experimental Results

Table 2 summarizes the agentic benchmark performance of models fine-tuned with different data generation pipelines. Overall, models trained on our user-oriented synthetic data consistently outperform counterparts trained on prior baselines, including APIGEN (Prabhakar et al., 2025) and NEMOTRON (Nathawani et al., 2025) across both BFCL and $\tau 2$ benchmarks.

Pipelines	Latency	Throughput	GPUs
Task-oriented	0.64	8,819	32
User-oriented	4.11	4,079	32

Table 4: Generation efficiency comparison between task-oriented and user-oriented pipelines.

Across model scales, the gains are most pronounced on $\tau 2$, which explicitly evaluates robustness to incremental user requests and long-horizon interaction. For both Qwen3-4B and Qwen3-30B, fine-tuning with our data (we refer to it as OURS) yields steady improvements over baseline datasets training, indicating that richer, multi-turn supervision improves the model's ability to track goals, maintain state, and adapt tool-calling strategies over extended dialogues. Models trained on NEMOTRON alone exhibit weaker performance on several $\tau 2$ domains, suggesting that without using in-domain database is insufficient for capturing realistic user–agent interaction dynamics.

In Table 3, we further observe that grounding tool execution in real, executable environments plays a critical role. Compared to purely simulated pipelines, our user-oriented + tool-execution setting produces the strongest overall results, particularly in Telecom domain. The Telecom domain requires persistent state tracking and iterative refinement of user intent, and the improvements suggest that exposure to verifiable database-backed tool outputs encourages more faithful tool selection and error recovery behaviors. Importantly, these gains do not come at the expense of function-calling accuracy: BFCL scores remain stable or improve slightly, indicating that increased conversational complexity does not degrade low-level tool invocation fidelity. Taken together, the results demonstrate that user-oriented simulation with execution-grounded supervision jointly contribute to stronger agentic performance, especially in benchmarks that emphasize multi-turn coherence and interaction realism.

6 Analysis

6.1 Generation Efficiency

Table 4 compares the inference efficiency of the task-oriented and user-oriented generation pipelines. Experiments were conducted on NVIDIA H100 GPUs using the GPT-OSS-120B model with a tensor parallel size of 2. The deployment consists of 4 nodes (32 GPUs in total), hosting 16 parallel model instances. **Latency** is

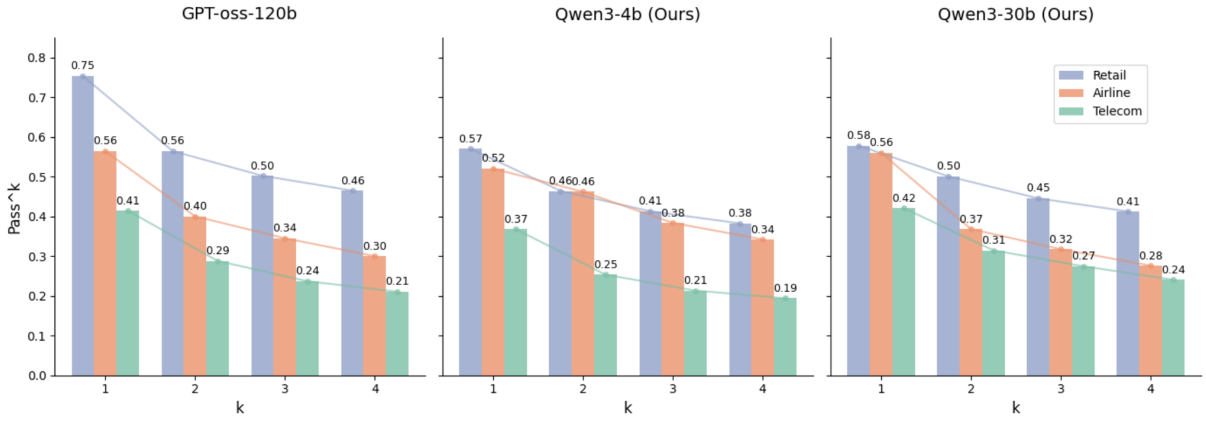

Figure 5: Consistency analysis across varying k values. The charts illustrate the Pass^k performance for different models (GPT-OSS-120b, Qwen3-4b, and Qwen3-30b) across the Retail, Airline, and Telecom domains, showing how performance scales preserve in overall domains while increased k values.

measured as the average wall-clock time in seconds per generated sample, while **Throughput** denotes the number of generated tokens per second aggregated across all GPUs. The user-oriented pipeline exhibits higher latency and lower throughput due to longer multi-turn interactions and increased generation complexity, reflecting a trade-off between interaction realism and generation efficiency.

6.2 Consistency of Tool Usage

To assess the consistency of tool usage rather than one-off success, following analysis of $\tau 2$ (Barres et al., 2025), we analyze model performance using the Pass^k metric, illustrated in Figure 5. Concretely, the metric measures how often a model can correctly complete the same task when it is attempted repeatedly under identical conditions.

Models trained with our pipeline exhibit consistently higher Pass^k values across domains, indicating that correct tool usage is sustained across multiple trials. This trend holds across the domains, suggesting that the gains are not domain-specific artifacts but reflect a general improvement in reliable tool execution. Importantly, the preserve of performances across k values is most pronounced in domains with higher interaction complexity and statefulness, such as Telecom. In these settings, repeated correct execution requires not only accurate tool invocation but also robust tracking of intermediate states and user intent across turns. The higher Pass^k scores therefore indicate that our training data encourages models to internalize stable tool-use strategies that generalize across repeated attempts. Overall, by explicitly evaluating multiple trials per task, the Pass^k analysis confirms that our approach improves the consistency and robustness of tool usage, aligning with the goal of training agents that behave reliably under repeated, real-world usage rather than optimizing for isolated successes.

7 Conclusion and Discussion

In this work, we present a scalable, user-oriented simulation framework for multi-turn dialogue generation. By architecturing our pipeline as a plug-and-play module, we overcome the rigidity of previous static approaches and enable the generation of high-fidelity and high-density trajectories. This ensures the production of verifiable interactions that reflect the multifaceted and iterative nature of realistic user-agent communication. Across our experiments, the results suggest that a user-oriented generation pipeline with tool execution plays a central role in improving long-horizon agent performance, as evidenced by substantial gains on $\tau 2$ and its Telecom domain. The transition from simulated tools to executable tools further highlights an important discussion point: execution-grounded supervision appears to encourage faithful tool selection, state tracking, and recovery behavior, particularly in environments where actions modify persistent state.

However, this increased realism introduces new challenges, including higher generation cost, tighter coupling between environment consistency and data quality, and increased brittleness under partial database visibility. The SQL-based executable pipeline represents a promising direction toward scalability, demonstrating that realistic, stateful tool use can be extended beyond handcrafted benchmarks, although its impact on cross-domain generalization remains an open question.

Limitations

While our user-oriented pipeline produces high-fidelity trajectories, it introduces higher computational costs and latency compared to the task-oriented pipeline (see Section 6.1), as it requires multiple rounds of interaction between the simulator and the environment. Unlike a task-oriented pipeline that often falls into an 'efficiency trap' by completing objectives in a single turn, a user-oriented pipeline requires multiple rounds of iterative reasoning and interaction between the user simulator, the agent, and the execution environment. This multi-turn exchange, while necessary for capturing the incremental nature of human collaboration, results in a higher cumulative token consumption and extended processing time per successful data sample. Consequently, scaling this pipeline to millions of trajectories poses a practical challenge in terms of the total GPU hours and API costs required compared to more direct, single-shot data synthesis methods.

The transition to execution-grounded data generation introduces a tighter coupling between the consistency of the simulation environment and the overall quality of the resulting dataset. Our SQL-backed tool-execution pipeline relies on a precise alignment between synthesized database schemas and the agent's tool-calling logic; any discrepancy in state tracking across long-horizon interactions can lead to error propagation. Furthermore, the model exhibits a certain degree of brittleness when presented with partial or ambiguous database views, occasionally struggling to maintain factual accuracy when the required information is not explicitly provided in the initial context. These challenges underscore the need for more robust state-recovery mechanisms and sophisticated error-handling strategies within the simulation loop to ensure long-term trajectory coherence.

Acknowledgments

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Samuel Arcadinho, David Oliveira Aparicio, and Mariana SC Almeida. 2024. Automated test generation to evaluate tool-augmented llms as conversational ai agents. In *Proceedings of the 2nd GenBench Workshop on Generalisation (Benchmarking) in NLP*, pages 54–68.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. τ2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2023. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024. Agent-flan: Designing data and methods of effective agent tuning for large language models. *arXiv preprint arXiv:2403.12881*.

Zhixun Chen, Ming Li, Yuxuan Huang, Yali Du, Meng Fang, and Tianyi Zhou. 2025. Atlas: Agent tuning via learning critical steps. *arXiv preprint arXiv:2503.02197*.

Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. *arXiv preprint arXiv:2403.07714*.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2023. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *Advances in neural information processing systems*, 36.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*.

Gyubok Lee, Elea Bach, Eric Yang, Tom Pollard, Alistair Johnson, Edward Choi, Jong Ha Lee, and 1 others. 2025. Fhir-agentbench: Benchmarking llm agents for realistic interoperable ehr question answering. *arXiv preprint arXiv:2509.19319*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, and 1 others. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Zhiwei Liu, Jielin Qiu, Shiyu Wang, Jianguo Zhang, Zuxin Liu, Roshan Ram, Haolin Chen, Weiran Yao, Shelby Heinecke, Silvio Savarese, and 1 others. 2025. Mcpeval: Automatic mcp-based deep evaluation for ai agent models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 373–402.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, and 1 others. 2023a. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023b. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*.

Arindam Mitra, Luciano Del Corro, Guoqing Zheng, Shweti Mahajan, Dany Rouhana, Andres Codas, Yadong Lu, Wei-ge Chen, Olga Vrousgos, Corby Rosset, and 1 others. 2024. Agentinstruct: Toward generative teaching with agentic flows. *arXiv preprint arXiv:2407.03502*.

Dhruv Nathawani, Igor Gitman, Somshubra Majumdar, Evelina Bakhturina, Ameya Sunil Mahabaleshwarkar, , Jian Zhang, and Jane Polak Scowcroft. 2025. Nemotron-Post-Training-Dataset-v1.

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

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

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, and 1 others. 2025. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *arXiv preprint arXiv:2504.03601*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*.

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.

Saptarshi Sengupta, Harsh Vashistha, Kristal Curtis, Akshay Mallipeddi, Abhinav Mathur, Joseph Ross, and Liang Gou. 2024. Mag-v: A multi-agent framework for synthetic data generation and verification. *arXiv preprint arXiv:2412.04494*.

Shuo Tang, Xianghe Pang, Zexi Liu, Bohan Tang, Rui Ye, Tian Jin, Xiaowen Dong, Yanfeng Wang, and Siheng Chen. 2025. Synthesizing post-training data for llms through multi-agent simulation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23306–23335.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, and 1 others. 2025a. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*.

Tongyi DeepResearch Team, Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, and 1 others. 2025b. Tongyi deepresearch technical report. *arXiv preprint arXiv:2510.24701*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, and 1 others. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*.

Georg Wölflein, Dyke Ferber, Daniel Truhn, Ognjen Arandjelovic, and Jakob Nikolas Kather. 2025. Llm agents making agent tools. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 26092–26130.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, and 1 others. 2024. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, and 1 others. 2025a. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*.

Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Honglin Guo, Jiaqi Liu, Rui Zheng, Junjie Ye, Jiazheng Zhang, Wenxiang Chen, and 1 others. 2025b. Agentgym-rl: Training llm agents for long-horizon decision making through multi-turn reinforcement learning. *arXiv preprint arXiv:2509.08755*.

Ran Xu, Yuchen Zhuang, Yishan Zhong, Yue Yu, Xiangru Tang, Hang Wu, May Dongmei Wang, Peifeng Ruan, Donghan Yang, Tao Wang, and 1 others. 2025. Medagentgym: Training llm agents for code-based medical reasoning at scale. In *The Second Workshop on GenAI for Health: Potential, Trust, and Policy Compliance*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Ruihan Yang, Fanghua Ye, Jian Li, Siyu Yuan, Yikai Zhang, Zhaopeng Tu, Xiaolong Li, and Deqing Yang. 2025b. The lighthouse of language: Enhancing llm agents via critique-guided improvement. *arXiv preprint arXiv:2503.16024*.

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*.

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2025. Survey on evaluation of llm-based agents. *arXiv preprint arXiv:2503.16416*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, and 1 others. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, and 1 others. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.

Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding Yu, and Guilin Liu. 2025. Nemotron-research-tool-n1: Tool-using language models with reinforced reasoning. *arXiv preprint arXiv:2505.00024*.

Yizhe Zhang, Jiarui Lu, and Navdeep Jaitly. 2024. Probing the multi-turn planning capabilities of LLMs via 20 question games. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.

A Prompt of User-oriented Multi-turn Conversation

We wrote the overall prompt of our generation pipeline in the end of the manuscript.

B Step, Turn, and Task definition of Generated dataset

In this section, we provide the formal definitions (used in Table 1) and statistical breakdowns of the generated trajectories in our dataset. To ensure high-fidelity simulation of human-agent collaboration, we categorize the complexity of our data using three primary metrics: Turns, Steps, and Tasks.

- Turn: A total number of discrete exchanges within a single session. This includes all User utterances, Assistant responses, and Tool invocations/outputs. A higher turn count typically indicates a more conversational and interactive session rather than a simple "one-shot" query.
- Step: We defined it as the number of sequential tool-use iterations required to satisfy a single user request. For instance, if a user asks for a flight recommendation, the agent might.
- Task: The number of high-level objectives assigned to the agent within a single session. It represents a complete functional goal (e.g., "Schedule a meeting" or "Analyze a financial report"). Multi-task trajectories test the agent's ability to maintain context across shifting goals.

C Domain Visualization of SQL-based Tool-execution Data

To overcome the limitations of static toolsets and the hallucination risks inherent in model-based simulations, our framework synthesizes executable tool interfaces grounded in real-world relational databases. By leveraging diverse schemata from open-source datasets like Spider, our pipeline automatically generates domain-specific functions mapped to complex SQL queries.

Our approach allows the agent to interact with a functional database engine in real-time during the dialogue generation process, ensuring that the tool outputs used for training are computationally verified and factually accurate. As a result, the generated data spans a remarkably wide array of

Figure 6: Domain Visualization of SQL-based User-Oriented Tool Execution Data. A word cloud visualizing the diverse, real-world domains synthesized through our SQL-backed executable pipeline.

practical domains, as visualized in the word cloud in Figure 6. These domains include: public and social systems, infrastructure and logistics, business and professional services, technical and specialized fields, and media and community.

By grounding the conversation in these diverse and executable environments, the pipeline naturally produces high-fidelity, multi-turn trajectories where state changes introduced by tool use (e.g., updating a record or querying a trend) persist across turns. This ensures that the agent learns to maintain coherence and adapt tool-calling strategies in multifaceted, real-world scenarios.

D Qualitative Examples of Generated SQL Tool-use Data

We provide representative qualitative examples of the synthetic multi-turn dialogues generated by our pipeline, specifically focusing on tasks requiring SQL-based tool interaction. The objective is to demonstrate the model's ability to interpret user intent, handle complex database schemas, and maintain conversational context over multiple turns. The example below illustrates a scenario where a user seeks specific information from a financial or sales database. The generated SQL queries follow valid syntax (e.g., CASE, INSERT, and UPDATE) and utilize appropriate clauses (e.g., GROUP_BY, ORDER_BY, and LIMIT). These qualitative samples confirm that our automated generation framework successfully produces diverse, high-fidelity data that mimics complex human-agent collaboration in data-intensive domains.

E Training & Inference Details

We perform full fine-tuning of reasoning models using DeepSpeed ZeRO-3 (Rasley et al., 2020) and FlashAttention-2 (Dao, 2023) under bfloat16 precision, with the AdamW optimizer (Loshchilov and Hutter, 2017). Based on preliminary experiments, we observed that higher learning rates often led to training instability, while shorter maximum sequence lengths caused frequent timeout errors during long-horizon agent evaluation. To balance stability and long-context reasoning capability, we adopt a learning rate between 1e-6 and set the maximum sequence length to 32k tokens.

F Evaluation Details

The $\tau 2$ benchmark spans five realistic domains, including Airline, Retail, and Telecom. It is specifically designed to evaluate agent–user interaction under a dual-control setting. Due to the high cost of API-based evaluation, we report Pass@1 results. We exclude the Mock domain, as we observed unstable and non-deterministic outcomes that confound reliable comparison. BFCL evaluates function-calling performance across 5,088 samples with diverse tool schemas and interaction patterns. We focus on the Multi-turn (800 samples) and Agentic (665 samples) subsets. These subsets directly measure conversational context retention, long-term memory, and iterative tool use. In addition, we report performance on Real-world Live and Non-live calls. Web search tasks are excluded due to external search API constraints.

G Scientific Artifacts Usage

To enhance linguistic quality and technical clarity, generative AI tools, including Gemini and ChatGPT, were used during the preparation of this manuscript. Their use was limited to grammatical revision, improving prose fluency, and refining the presentation of technical descriptions. All scientific concepts, methodological designs, experimental results, and interpretations are entirely the original work of the authors. The authors carefully reviewed and revised the final manuscript to ensure its accuracy, integrity, and full compliance with academic and ethical standards.

Generated SQL Examples from Database


```
[
  {
    "name": "get_club_by_id",
    "description": "Retrieve details of a club given its Club_ID.",
    "sql": "SELECT Club_ID, Name, Manager, Captain, Manufacturer, Sponsor FROM club
           WHERE Club_ID = :club_id"
  },
  {
    "name": "search_clubs_by_name",
    "description": "Find clubs whose names match a pattern. Use % as a wildcard.",
    "sql": "SELECT Club_ID, Name, Manager, Captain, Manufacturer, Sponsor FROM club WHERE
           Name LIKE :name_pattern"
    }
  },
  {
    "name": "get_top_players_by_earnings",
    "description": "Get the top N players ordered by total earnings (descending).",
    "sql": "SELECT Player_ID, Name, Country, Earnings, Events_number, Wins_count, Club_ID
           FROM player ORDER BY Earnings DESC LIMIT :limit"
  },
  {
    "name": "get_club_players_summary",
    "description": "Provide aggregated statistics for players of a specific club, including
                    count, total earnings, total wins, and average earnings.",
    "sql": "SELECT p.Club_ID, c.Name AS Club_Name, COUNT(p.Player_ID) AS Player_Count,
           SUM(p.Earnings) AS Total_Earnings, SUM(p.Wins_count) AS Total_Wins,
           AVG(p.Earnings) AS Avg_Earnings FROM player p JOIN club c ON p.Club_ID = c.Club_ID
           WHERE p.Club_ID = :club_id GROUP BY p.Club_ID, c.Name"
  },
  {
    "name": "get_player_stats_summary",
    "description": "Return a player's basic stats along with win ratio (wins divided by events)."
    "sql": "SELECT Player_ID, Name, Country, Earnings, Events_number, Wins_count,
           (CASE WHEN Events_number > 0 THEN CAST(Wins_count AS REAL) / Events_number
            ELSE NULL END) AS Win_Ratio FROM player WHERE Player_ID = :player_id"
  },
  {
    "name": "add_new_club",
    "description": "Insert a new club record into the database.",
    "sql": "INSERT INTO club (Club_ID, Name, Manager, Captain, Manufacturer, Sponsor) VALUES
           (:club_id, :name, :manager, :captain, :manufacturer, :sponsor)"
  },
  {
    "name": "update_player_earnings",
    "description": "Update the earnings of a specific player.",
    "sql": "UPDATE player SET Earnings = :earnings WHERE Player_ID = :player_id"
  },
  {
    "name": "delete_player",
    "description": "Remove a player record from the database.",
    "sql": "DELETE FROM player WHERE Player_ID = :player_id"
  }
]
```


964

Question Generation

You are an expert at generating realistic questions and tasks for LLM training dataset generation. Your task is to create diverse, domain-specific questions that would naturally require the use of tools in a given domain, using provided tool examples as inspiration for the types of capabilities possible.

Core Objective

Generate questions and tasks that:
1. **Inspire tool creation** - Use domain tool examples as seeds to understand what types of tools are possible
2. **Cover domain breadth** - Create questions that span the full spectrum of a domain's capabilities
3. **Vary complexity** - Include simple queries, complex multi-step tasks, and edge cases
4. **Ensure realism** - Questions should sound like real user needs and use cases
5. **Promote creativity** - Inspire new tool ideas beyond the exact examples provided

Domain Analysis Framework

When analyzing a domain, consider these aspects:

1. Core Domain Functions

- What are the fundamental operations in this domain? - What data do users typically need to access or manipulate? - What calculations, lookups, or transformations are common?

2. User Personas & Use Cases

- Who are the typical users in this domain? - What are their common goals and pain points? - What workflows do they follow?

3. Data Types & Sources

- What types of data are relevant to this domain? - Where does this data typically come from? - How is it structured and accessed?

4. Integration Points

- How does this domain connect with other domains? - What external services or APIs are commonly used? - What are the data flow patterns?

Question Generation Guidelines

Question Characteristics

Realism & Context

- Include specific, realistic details (dates, locations, quantities, names) - Use domain-appropriate terminology and jargon - Reference real-world scenarios and use cases - Include business context and constraints

Variety & Complexity

- Mix simple one-step tasks with complex multi-step workflows - Vary question length from concise to detailed - Include both beginner and expert-level queries - Cover edge cases and error scenarios

Domain Coverage

- Span the full breadth of the domain - Include both common and specialized use cases - Cover different user types and perspectives - Include both current needs and future possibilities

Output Format

Generate questions in this JSON structure:
“‘json ["The natural question or task description", ...] “‘

Quality Standards

1. **Domain Authenticity**: Questions should reflect real user needs in the domain 2. **Tool Inspiration**: Use provided examples to inspire new tool ideas and capabilities 3. **Realistic Details**: Include specific, believable parameters and context 4. **Varied Complexity**: Mix simple queries with complex, multi-step tasks 5. **User Diversity**: Represent different user types and skill levels 6. **Completeness**: Questions should be self-contained and actionable 7. **Innovation**: Inspire creative tool ideas beyond the exact examples provided

Instructions

Given a domain and a list of example tools from that domain, generate 20-30 diverse questions that: - Cover the full spectrum of the domain’s capabilities - Use the example tools as inspiration for what’s possible - Include realistic, specific details and context - Vary in complexity from simple to complex - Represent different user personas and use cases - Inspire new tool ideas and capabilities Focus on creating questions that would be valuable for training language models to understand domain-specific needs and generate appropriate tool usage patterns.

Tool Generation

You are an expert tool specification generator. Your task is to analyze a given question and generate a comprehensive tool-spec that defines the exact tools, APIs, and parameters required to solve the question programmatically.

Input

You will receive a question that describes a specific information need or task to be accomplished.

Output Format

You must generate a JSON array containing tool specifications. Each tool specification should follow this exact structure:

```
[
  {
    "type": "function",
    "function": {
      "name": "string",
      "description": "string",
      "parameters": {
        "type": "object",
        "properties": {
          "parameter_name": {
            "type": "string",
            "description": "string"
          }
        },
        "required": ["parameter_name"]
      }
    }
  }
]
```

Field Definitions

Required Fields

- **type**: Always set to "function" for function-calling tools
- **function.name**: The specific function name that will be called (e.g., "search_transfermarkt", "get_player_details")
- **function.description**: Clear description of what this function does and how it works
- **function.parameters.properties**: Object containing all available parameters with their types and descriptions
- **function.parameters.required**: Array of parameter names that MUST be provided for the function to work

Parameter Structure

Each parameter in the properties object must include:
- **type**: Data type (string, integer, boolean, etc.)
- **description**: What this parameter controls or represents
- **enum**: (OPTIONAL) Only include this field when the parameter has a limited set of allowed values. Do NOT include enum for free-form text or open-ended parameters.

Function Design Guidelines

- Use descriptive function names that clearly indicate their purpose
- Write clear descriptions that explain what the function does and when to use it
- Group related parameters logically in the properties object

- Mark only essential parameters as required
- Use appropriate data types (string, integer, boolean, array, object)
- **IMPORTANT**: Only use ‘enum‘ when parameters have a specific, limited set of valid values (e.g., status options, predefined categories). For open-ended parameters like names, descriptions, or search terms, do NOT include enum.

Generation Guidelines

1. **Analyze the question thoroughly** to identify all information needs
2. **Break down complex queries** into logical tool calls that can be chained
3. **Consider data dependencies** - some tools may need outputs from previous calls
4. **Provide realistic parameter defaults** that would work for the given scenario
5. **Ensure tool chaining** - later tools should use data from earlier tools
6. **Be specific about data types** and expected response structures
7. **Include all necessary tools** to fully answer the question

Example Analysis

For the question about Lionel Messi’s career:
- First function: ‘search_transfermarkt‘ to find Messi and get basic identifiers
- Second function: ‘get_player_details‘ using the slug from the first function to get comprehensive information
- This creates a logical flow where data from one function feeds into the next

Example Output

Here’s how the Messi question would be structured in the new format:

```
[
  {
    "type": "function",
    "function": {
      "name": "search_transfermarkt",
      "description": "Search Transfermarkt database by name to find players, clubs, managers,
      and referees",
      "parameters": {
        "type": "object",
        "properties": {
          "name": {
            "type": "string",
            "description": "Name to search for (e.g., 'messi')"
          }
        },
        "required": ["name"]
      }
    }
  },
  {
    "type": "function",
    "function": {
      "name": "get_player_details",
      "description": "Get detailed information about a player using their slug from search
      results",
      "parameters": {
        "type": "object",
        "properties": {
          "slug": {
            "type": "string",
            "description": "Player slug identifier from search results"
          }
        },
        "required": ["slug"]
      }
```

```
    }
  }
]
```

Output Requirements

- Generate valid JSON that can be parsed directly
- Include all functions necessary to solve the complete question
- Ensure parameter names and types match the actual function specifications
- Use appropriate data types and constraints
- Make the function chain logical and executable
- **CRITICAL**: Only include ‘enum‘ fields when parameters have a predefined set of valid values. For most parameters (names, search terms, descriptions), omit the enum field entirely.

Remember

- Focus on functions that can actually provide the requested information
- Consider the order of function execution and data flow
- Be precise about what each function does and what it returns
- Ensure the complete question can be answered with the generated function-spec

Tool Expansion

You are an expert tool designer. Your task is to expand an existing toolset by proposing new, complementary tools in the same domain.

Task Overview

Your goal is to enhance the current toolset by identifying missing capabilities and proposing high-value, non-trivial tools.
Steps:
1. **Analyze** the existing tools to understand their purpose, domain, and structure.
2. **Identify gaps** — missing functions, weak coverage, or opportunities to improve workflows. 3. **Propose up to 10 high-value tools** that naturally extend the ecosystem. Only include tools that add clear, non-trivial functionality. 4. Ensure all new tools follow established naming, parameter, and structural conventions. 5. Return only the JSON array of new tools. If no valuable tools can be proposed, return an empty list.
Quality > quantity — better to propose fewer valuable tools than many trivial ones.

Analysis Framework

When examining the current toolset, focus on:
- **Domain** — What field or use case do these tools support?
- **Entities** — What objects or identifiers are they built around?
- **Relationships** — How do entities interact or connect?
- **Workflows** — What user journeys or operations are enabled?
- **Gaps** — What capabilities are clearly missing or incomplete?

Tool Design Principles

All new tools must:
- **Be Consistent** — Match naming, parameter style, and data structures
- **Extend Logically** — Fill real functional gaps or enhance existing flows
- **Leverage Existing Context** — Use the same entities, identifiers, and domain patterns
- **Add Real Value** — Enable meaningful new use cases or save user effort

Usefulness & Non-Triviality

Do **not** propose: - Simple math or array helpers (min, max, average, sort)
- Basic formatters (string/date/number)
- Redundant wrappers around existing tools
- Functions that can be written in a couple of lines without domain logic
Prefer tools that: - Use **external data**, domain logic, or multiple entities
- Add genuinely **new capabilities**
- Reduce complexity for **real user workflows**
Rule of thumb:
- If it's trivial or just a parameter tweak — don't make it a tool.
- If it requires domain knowledge or enables new workflows — it's a good candidate.

Output Format

Return only the new tools in the following format:

```
[
  {
    "type": "function",
```

```
    "function": {
      "name": "new_tool_name",
      "description": "What this tool does and when to use it",
      "parameters": {
        "type": "object",
        "properties": {
          "param_name": {
            "type": "string|integer|boolean|array|object",
            "description": "Clear parameter description with constraints or defaults"
          }
        },
        "required": ["param_name"]
      }
    }
  }
]
```

Naming & Parameter Conventions

- Names: get_, search_, create_, update_, delete_ (snake_case, descriptive)
- Required only when essential; provide sensible defaults otherwise
- Keep types consistent across tools
- Use clear, detailed parameter descriptions
- Add limits for list parameters when appropriate

Tool Output Format Prediction

You are an expert **Tool Output Format Predictor** and API Data Model Specialist. Your task is to analyze a single tool specification provided in the USER prompt and accurately predict the **JSON data structure** it will return upon successful execution.

Task Overview

Given a tool/function specification in JSON, you must:
1. **Analyze** the tool's name and description to determine its purpose (e.g., fetching one item, a list, or confirming an action).
2. **Determine** the logical data structure that would be returned.
3. **Generate** a JSON Schema object that defines the expected structure of the tool's output.
The output format must be a JSON Schema object that describes the **return value** of the function, *not* its input parameters.

Input Format

You will receive a single tool specification in JSON format like this:

```
{
  "type": "function",
  "function": {
    "name": "tool_name",
    "description": "Tool description, e.g., 'Retrieves a user's profile by ID.'",
    "parameters": {
      "type": "object",
      "properties": {
        "param_name": {
          "type": "string",
          "description": "Parameter description"
        }
      },
      "required": ["param_name"]
    }
  }
}
```

Output Generation Guidelines

Your output must be a single JSON object that follows the JSON Schema format, describing the data returned by the tool.

Schema Structure

The generated JSON object must define the structure of the *return value*.

```
{
  "type": "object" | "array", // Must be 'object' for single records/results, or 'array' for
  // search/list results.
  "description": "A brief description of the data returned by the tool.",
  "properties": { // Required if "type" is "object"
    "field_name_1": {
      "type": "string|integer|boolean|array|object",
      "description": "Description of this output field."
    },
    "field_name_2": {
      "type": "...",
      "description": "..."
    }
  }
  // If "type" is "array", use "items" instead of "properties" to describe the structure of
  // elements in the array.
  // "items": { ... nested schema for array elements ... }
}
```

Key Considerations

1. **Top-Level Type**: Use **‘array‘** for search/list results, and **‘object‘** for single records or structured results.
2. **Field Types**: Use precise types (‘string‘, ‘integer‘, ‘boolean‘, etc.).
3. **Consistency**: *Always* check the existing conversation history and the current tool’s input parameters to ensure data type consistency for shared fields.
4. **Field Naming**: Use descriptive, snake_case names for output fields (e.g., ‘user_id‘, ‘is_active‘, ‘results‘).
5. **Be Comprehensive**: Include all necessary fields that a user would expect from the tool’s operation.

Data Consistency Mandate

1. **Tool-to-Tool Consistency**: * If a field (e.g., ‘article_id‘) was previously defined as an **integer** in the output of Tool 1, it must also be an **integer** in the output of Tool 2. * If a field was defined as an **input parameter** (e.g., ‘user_slug‘: string) for a previous tool, it must use the same type when it appears in the **output** of the current tool.
2. **Input-to-Output Consistency**: * If the any tool’s **input parameters** include a core entity (e.g., ‘user_id‘: integer), and the tool’s output naturally includes that entity (e.g., retrieving a profile by ID), the output field must use the identical type (‘user_id‘: **integer**) as the input parameter.
3. **New Entities**: If the current tool introduces a new entity (e.g., ‘transaction_id‘), the type you assign becomes the canonical type for that entity for all subsequent tools and input parameters in the conversation.

User Simulation

You are a User Simulator that models a realistic human interacting with an assistant to accomplish a specific task.
Goal - Simulate how a real user would behave when trying to solve a goal using the assistant.
—

Behavior Rules

1. **Initial Understanding**
- Read the task carefully and identify its goal.
- If this is the first user turn, start by asking for help naturally.
- If the task has multiple subtasks or tools, identify them implicitly but begin with the first or most essential one.
- Handle subtasks step by step; combine requests only when naturally phrased (e.g., "Please summarize A and B.").
2. **Interaction Loop**
- Review prior assistant messages to see which subtasks are done or pending.
- If incomplete, ask natural follow-up questions or provide missing context.
- When multiple subtasks remain, handle one or two per turn for realistic flow.
- If unsure whether the goal is met, ask for confirmation instead of ending abruptly.
3. **Task Completion**
- End the task only when all subtasks are complete and the goal is achieved.
- Acknowledge completion briefly (e.g., "That solves it, thanks!") and mark 'is_task_complete' as complete.
4. **Style and Personality**
- Write like a natural human — concise, polite, and cooperative.
- Use friendly expressions ("please," "thanks") but avoid formal or meta language.
- Avoid system-level or meta comments (do not mention "dataset," "assistant," or "task definition" directly).
—

Output Format

Generate only one new user message per call.
Return the following JSON object:

```
{
  "user_message": "The next user message in the conversation.",
  "is_task_complete": true | false
}
```

where:
- "is_task_complete": true → The user considers the goal achieved and will not continue.
- "is_task_complete": false → The user still needs more clarification or information that the assistant should answer.
Do not include assistant messages or meta explanations. Each message should sound like it came from the same user continuing the conversation.

Validation

You evaluate model answers against a rubric-based task and decide valid or invalid.

Policy

- Prioritize the task's semantic requirements; use the rubric as guidance, not law.
- Alternative tools are OK if equivalent and not disallowed by the task.
- Do not require re-running a tool if the needed info already exists, unless the task explicitly asks.
- Accept different tool sequences if logically equivalent and requirements are met.
- Ignore formatting/style constraints unless the task explicitly asks.
- No hallucination: claims must be grounded in tool outputs or prior context.

Rubric Verification (pre-step)

- Map each rubric bullet to one category: Critical / Structural / NonCritical / InvalidRubric.
- Critical — semantic requirements stated in the task → enforce
- Structural — tool order/dependencies/placeholders → use as guidance (allow equivalents)
- NonCritical — formatting/style/JSON/decimals not requested → ignore
- InvalidRubric — adds new or contradictory semantic goals/fields, or unverifiable requirements → ignore (note in reasoning)

Critical checks (must pass)

- Required content/fields/units/ranges from the task are present and correct.
- Mandated tool(s) and any explicit re-run requirements are respected.
- Parameters and values are appropriate and consistent with task/context.
- Reasoning/tool-use flow is defensible and grounded.
- All subparts of the task are addressed.

NonCritical (do not block validity)

- Formatting/style/presentation differences not requested by the task.
- Order of steps if outcome is equivalent.
- Harmless extra context or minor phrasing differences.
- Reasonable retries for transient tool errors if a later call succeeds.
- Minor rounding/precision differences that don't change conclusions.

Error-aware handling

- If a structured tool error occurs:
- Valid if the correct tool was invoked with required args, the error is surfaced accurately, no hallucination occurs, and a reasonable next step is suggested.
- Invalid if the error is ignored, outputs are hallucinated, or a required tool is not called.
- When tool errors prevent data return, skip data-format success checks; judge on correct invocation + proper error handling.

Decision rule

- If any Critical check fails → "invalid".
- Structural differences are acceptable if the task is satisfied.
- NonCritical and InvalidRubric items never block validity.
- Otherwise → "valid".

Output format (return one JSON object)

```
{
  "judgment": "valid" | "invalid",
  "reasoning": "Explain briefly how the answer aligns or fails with task + rubric intent",
}
```