

Unlocking Implicit Experience: Synthesizing Tool-Use Trajectories from Text

Zhihao Xu^{1,2,3*†}, Rumei Li^{3*}, Jiahuan Li³,

Rongxiang Weng^{3‡}, Jingang Wang^{3‡}, Xunliang Cai³, Xiting Wang^{1,2}

¹Gaoling School of Artificial Intelligence, Renmin University of China Beijing, China

²Beijing Key Laboratory of Research on Large Models and Intelligent Governance

³Meituan, China

zhihaoxu@ruc.edu.cn

Abstract

Enabling Large Language Models (LLMs) to effectively utilize tools in multi-turn interactions is essential for building capable autonomous agents. However, acquiring diverse and realistic multi-turn tool-use data remains a significant challenge. In this work, we propose a novel text-based paradigm. We observe that textual corpora naturally contain rich, multi-step problem-solving experiences, which can serve as an untapped, scalable, and authentic data source for multi-turn tool-use tasks. Based on this insight, we introduce GEM, a data synthesis pipeline that enables the generation and extraction of multi-turn tool-use trajectories from text corpora through a four-stage process: relevance filtering, workflow & tool extraction, trajectory grounding, and complexity refinement. To reduce the computational cost, we further train a specialized Trajectory Synthesizer via supervised fine-tuning. This model distills the complex generation pipeline into an efficient, end-to-end trajectory generator. Experiments demonstrate that our GEM-32B achieves a 14.9% improvement on the BFCL V3 Multi-turn benchmark. Our models partially surpass the performance of models trained on τ -bench (Airline and Retail) in-domain data, highlighting the superior generalization capability derived from our text-based synthesis paradigm. Notably, our Trajectory Synthesizer matches the quality of the full pipeline while significantly reducing inference latency and costs.

1 Introduction

The pursuit of Artificial General Intelligence (AGI) relies on the development of autonomous agents capable of perceiving, reasoning, and acting in complex real-world environments (Team et al., 2025; Liu et al., 2025; Su et al., 2025; Li et al., 2026). Such agents are required to dynamically utilize

*Equal Contributions.

†Work done during internship at Meituan LongCat Team.

‡Corresponding authors.

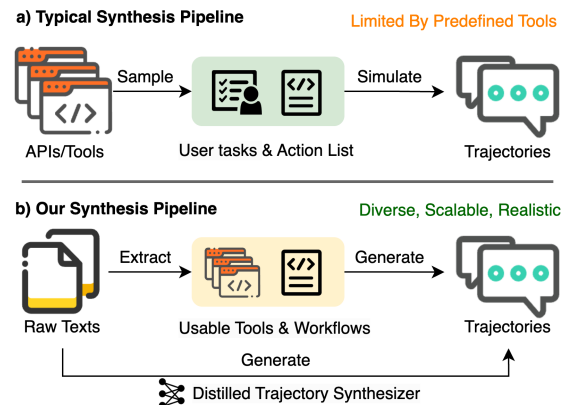


Figure 1: We propose the "Text to Trajectory" paradigm, instead of relying on predefined tools to generate tasks and trajectories.

diverse tools to extend their capabilities and accomplish multi-step tasks. While large language models (LLMs) have demonstrated impressive tool-use capabilities, they still struggle in realistic multi-turn interactions, particularly when faced with ambiguous instructions, long-context dependencies, and unexpected errors (Patil et al., 2025). These limitations constrain their practical application in agent-based systems (Yao et al., 2024; Barres et al., 2025; Zhang et al., 2025).

The primary bottleneck in training autonomous agents lies in the scarcity of high-quality, multi-turn tool-use trajectories, which are rarely found in real-world scenarios. To overcome this data scarcity, current research favors a tool-centered simulation paradigm (Fig. 1). These methods typically rely on predefined API sets to synthesize user tasks and simulate interactions (Qin et al., 2023; Liu et al., 2024b; Guo et al., 2024; Zeng et al., 2025; Prabhakar et al., 2025; Liu et al., 2024a; Yin et al., 2025; Yao et al., 2026; Jian et al., 2026). However, gathering a sufficiently diverse and comprehensive tool set is inherently expensive and difficult. The resulting tool-use training data is often limited by

the scope of the predefined APIs, failing to capture the complexity of real-world scenarios. This raises a critical question: *Can we bypass the dependency on predefined tools and synthesize diverse, high-quality trajectories directly from the real world?*

In this paper, we answer this question by proposing a novel text-based extraction paradigm. We observe that the raw text corpora used for pre-training large language models inherently contain multi-turn tool-use patterns. Although such texts do not contain explicit agentic trajectories, they often document rich, real-world multi-step problem-solving experiences, which can be extracted and transformed into multi-turn tool-use data (e.g., "the procedure for hospital reimbursement claims"). Our preliminary analysis of Ultra-fineweb (Wang et al., 2025) confirms that such texts contain actionable logical sequences spanning diverse domains, indicating that unstructured text serves as an untapped, scalable, and authentic source for synthesizing agentic training data.

To operationalize this paradigm, we introduce GEM, a data synthesis pipeline that enables the generation and extraction of multi-turn tool-use trajectories from text corpora. The GEM pipeline proceeds in four stages: (1) **Selection**, identifying text segments rich in multi-step workflows; (2) **Extraction**, deriving structured workflows and tool definitions to map human-described procedures to agent actions; (3) **Generation**, employing a powerful model to convert text and abstract workflows into concrete user-agent interactions; and (4) **Refinement**, enhancing the complexity and diversity of the trajectories followed by rigorous verification.

Extensive experiments validate the effectiveness of our approach. When fine-tuned on data synthesized by our pipeline, the GEM-32B model achieves a 14.9% improvement on BFCL V3 Multi-Turn benchmark. Notably, we find that out-of-domain training data generated through our paradigm achieves performance on τ^2 -bench that is comparable to models trained on τ -bench in-domain data, thereby demonstrating the strong generalization capability of our approach. Building upon this high-quality data, we further develop a specialized Trajectory Synthesizer to internalize the "text-to-trajectory" mapping. This synthesizer provides a cost-effective, end-to-end solution for large-scale data generation, matching the quality of the multi-stage synthesis pipeline while significantly reducing costs.

In summary, our contributions are threefold:

- We propose a new paradigm for agent trajectory synthesis that directly extracts multi-turn tool-use trajectories from text corpora. This paradigm unlocks an untapped, scalable source of authentic human problem-solving behaviors for training autonomous agents.
- We develop GEM, a data synthesis pipeline that transforms raw text into multi-turn tool-use trajectories to prove the effectiveness of our proposed paradigm. We use GEM to synthesize a high-quality dataset that leads to significant performance gains: 13.8% on τ^2 -bench and a 14.9% on BFCL V3.
- To enable cost-effective and scalable data generation, we develop a specialized Trajectory Synthesizer via supervised fine-tuning. We demonstrate that this model successfully distills the full GEM pipeline into an end-to-end generator, matching the original synthesis quality while significantly reducing costs.

2 Problem Formulation

In this paper, we address the challenge of acquiring general multi-turn tool-use agent trajectories. We propose a novel paradigm that synthesizes such trajectories by extracting tool-use workflows directly from large-scale textual corpora. This approach naturally enjoys these advantages: the scale of real-world text ensures scalability, its inherent diversity covers multiple domains, and its foundation in grounded human problem-solving experiences yields high-quality, realistic agent data. Formally, we define the paradigm as:

- **Input:** Let \mathcal{C} denote a large-scale text corpus, where each element $c_i \in \mathcal{C}$ is a raw, unstructured text segment (e.g., a document or narrative) that contains multi-step workflows.
- **Output:** A complete list of tools $\mathcal{P} = \{p_1, \dots, p_m\}$, where m denotes the number of available tools and each p_i represents a tool defined in the standard OpenAI format. Additionally, a structured multi-turn tool-use trajectory $T = \{s, (u_1, a_1, o_1), \dots, (u_n, a_n, o_n)\}$ is produced, where s denotes the system prompt and at each turn t , u_t corresponds to a user's query or request, a_t denotes the assistant's natural language response or tool call, and o_t is the resulting observation from the tool's execution.

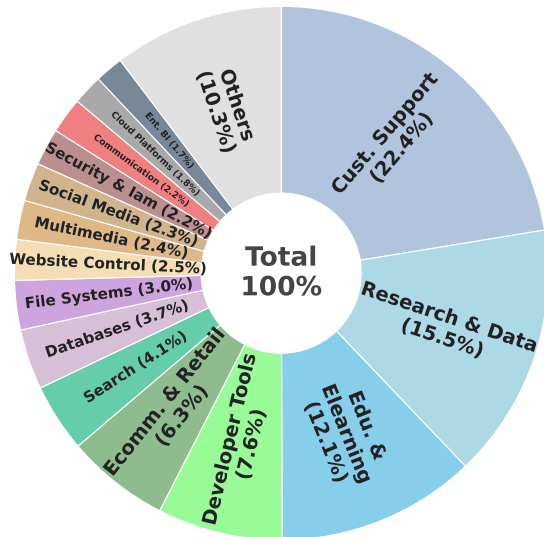


Figure 2: Distribution of the extracted task category from raw documents, showing the diversity of tasks covered in the text corpora.

Creating a Music Visualizer in Adobe After Effects

1. Prepare your audio file:

Before starting with After Effects, make sure you have an MP3 or WAV file of the song that you want to create a visualizer for.

2. Create a new composition:

Launch Adobe After Effects, then click 'Composition' > 'New Composition'. Set the width to 1920 pixels (or any other size depending on your project requirements) and set the height to 1080 pixels (for Full HD). Choose a frame rate - usually, 30fps works well but if you plan on exporting at higher quality like 60fps, choose this setting instead. Name your composition appropriately.

3. Import the Audio File:

Click 'File' > 'Import', select your audio file and hit 'Open'. Now drag and drop the imported audio file into your timeline within the Project panel.

User Query
Environmental Tools
Multi-step Workflows

Figure 3: Case study of "text to trajectory".

3 Methodology

In this section, we introduce GEM, an agentic synthesis pipeline designed to automatically extract and generate multi-turn tool-use trajectories directly from large-scale text corpora. We first conduct a preliminary analysis of text corpora in Section 3.1. Subsequently, we detail our trajectory collection pipeline in Section 3.2.

3.1 Preliminary Analysis

Setup To assess the feasibility of our proposed paradigm, we first perform a preliminary analysis on the characteristics of large-scale unstructured data, using the Ultra-fineweb corpus (Wang et al., 2025) as our main data source. We randomly sample approximately 250,000 raw text segments. Each segment is then processed through a sequential labeling pipeline: we first use a classifier to determine whether the text contains multi-step op-

erational procedures, which helps us identify segments suitable for conversion into tool-use trajectories. Only those segments identified as containing procedural content are subsequently annotated with rich metadata, including platform, domain, and task category, via Qwen3-8B to evaluate the diversity of the corpus. We also conduct a preliminary case study. The prompt templates used for metadata annotation are provided in Appendix A.1.

Results First, by observing a large number of text cases, we find that unstructured textual data inherently contain the three core components essential for constructing agentic trajectories as shown in Figure 3: (1) **User Queries**, which emerge naturally as goals or problems stated in the text; (2) **Environmental Tools**, whose descriptions, APIs, or functionalities are often embedded within explanatory or instructional contexts; and (3) **Multi-step Workflows**, manifesting as step-by-step procedures or operational narratives.

We find that around 14% of the sampled segments contain explicit multi-step workflows, indicating a substantial reservoir of procedural knowledge. Given the massive scale of available text corpora, this represents a significant potential source for generating diverse agent trajectories. Furthermore, the identified procedural segments cover a wide spectrum of task categories and application scenarios—including education & E-learning, data analysis as illustrated in Figure 2. This diversity confirms that textual sources can provide the necessary variety in tasks, tools, and environments required for robust agent training. The detailed domain distribution is further reported in Appendix D.

These findings collectively demonstrate that generating multi-turn tool-use trajectories directly from unstructured text is not only feasible but also taps into a vast and largely underexplored repository of agentic trajectories.

3.2 GEM Synthesis Pipeline

We now detail the data synthesis pipeline of GEM to generate and extract multi-turn tool-use data. The overall pipeline is illustrated in Figure 4. All prompts used in this section are in Appendix A.

Stage 1: Text Filtering To ensure the quality and realism of generated trajectories, the initial stage filters out raw text segments c that do not describe multi-step operations. The filtering procedure uses the same annotation prompt and model as described in Section 3.1.

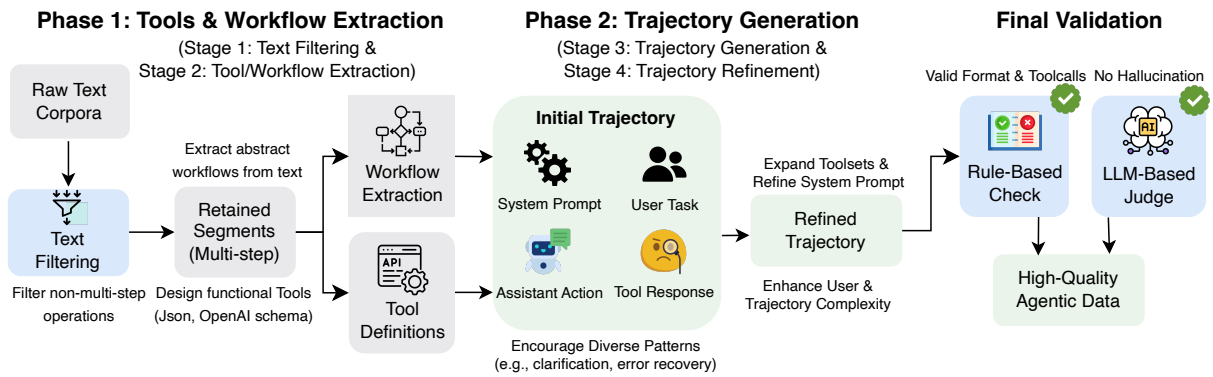


Figure 4: Overall Pipeline of GEM. The first phase extracts multi-step workflows from filtered text corpora and defines corresponding tools. The second phase generates and refines trajectories, incorporating diverse behavioral patterns and rigorous validation to produce high-quality training data.

Stage 2: Workflow & Tool Extraction For each retained text segment c containing multi-step operations, we proceed to extract structured abstract workflows and synthesize corresponding functional tools directly based on the text segment description. First, the model is instructed to identify all workflows and enumerate the individual steps within each (e.g., `search_items` and then `edit_item`). Importantly, the model is encouraged to recognize workflow complexity, including sequential dependencies, conditional logic, and uniqueness constraints, thereby enhancing the richness and practical relevance of the output. Concurrently, the model designs a set of API tools \mathcal{P} in accordance with OpenAI schema standards to support the subsequent trajectory generation. Each tool p is designed to perform a single, coherent function, with self-explanatory parameter names and well-specified data types. The final structured output comprises the abstract workflow descriptions alongside the complete tool definitions.

Stage 3: Trajectory Generation Based on the text, abstract workflows and tools synthesized in the previous stage, we now proceed to generate concrete multi-turn tool-use trajectories. For each workflow and its corresponding set of tools, we employ a strong teacher model (GLM-4.6 in this work due to its strong tool-use capability and open-source accessibility) to generate a preliminary complete trajectory T . Our approach synthesizes the full trajectory directly in a single pass instead of simulating turn-by-turn conversations via a multi-agent system to ensure efficiency. Each generated trajectory include the following core components:

- **System Prompt** s : Clear domain-specific rules extracted from the source text, estab-

lishing guidelines the assistant must follow throughout the conversation.

- **User Task** (u_1, \dots, u_n): A series of natural user requests that may be ambiguous or complex, reflecting real-world scenarios.
- **Assistant Responses** (a_1, \dots, a_n): Demonstrations of intelligent problem-solving, strict adherence to domain rules, and correct tool invocation with appropriate parameters.
- **Tool Response** (o_1, \dots, o_n): Simulated tool outputs that are both complete and realistic, providing authentic feedback.

Here, n denotes the number of conversational turns for each role. To capture the diverse practical challenges inherent in real-world multi-turn interactions, we encourage the inclusion of various interaction patterns during generation. These include but are not limited to: refusing requests that exceed the assistant’s capabilities, clarifying ambiguous user queries, and actively recovering from errors.

Stage 4: Refinement We observe that although the initial multi-turn dialogue trajectories T are complete, they often lack sufficient complexity and tend to be relatively straightforward. To fully utilize and enhance the quality of these trajectories, we adopt a refinement strategy to further process them. Specifically, given T , we generate a refined trajectory T' by expanding the variety of tools used, improving the realism of environmental responses, increasing the ambiguity and complexity of user requests, and ensuring the inclusion of non-trivial tool-call chains. We find this refinement essential for obtaining higher-quality agentic data, as demonstrated in Section 4.5.

Validation The trajectory filtering process integrates rule-based verification with LLM-based assessments to ensure high-quality outputs. Given a refined trajectory T' , we first apply a rule-based check to guarantee structural correctness. This involves verifying that all tools are correctly defined according to the OpenAI format and that each tool call corresponds to a valid function within the designated toolset, with argument names and types matching their definitions. We further validate the conversation format to confirm that tool responses meet all requirements and that all role tags are properly closed. Beyond structural correctness, we employ an LLM-based judge (Qwen3-32B in this paper) to detect and eliminate hallucinations. We use a 32B model for the judge because of verification-generation asymmetry, where verifying a solution is far easier than generating one. This step specifically examines all tool calls to ensure that every generated parameter value is explicitly grounded in the dialogue context rather than being fabricated. Only those trajectories that successfully pass both validation stages are retained as the final set T_{final} for use in supervised fine-tuning (SFT).

Overall, approximately 30% of trajectories are filtered by rule-based checks and 10% by LLM-based checks due to hallucinations.

Data Synthesizer Generating such trajectories is costly and time-consuming. To address this, we propose training a data synthesizer via supervised fine-tuning (SFT), which learns an end-to-end mapping from text to multi-turn tool-use trajectories. This approach allows for the cost-effective synthetic training data. For each data instance, the synthesizer takes as input x an instruction ("Turn the following text into multi-turn tool-use trajectories") paired with a text segment. It then generates the corresponding output y , which includes both the necessary tool definitions and the resulting multi-turn tool-use trajectory.

4 Experiments

4.1 Experimental Setups

Benchmark. We evaluate our approach on two challenging benchmarks designed to assess multi-turn tool-use capabilities: (1) BFCL V3 (Patil et al., 2025): This benchmark involves agents interacting with a Python-based API environment. We focus specifically on the multi-turn scenarios, which are divided into four categories: *Multi Turn Base*, *Miss Param*, *Miss Func*, and *Long Context*

with 200 tasks per category. (2) τ^2 -bench (Barres et al., 2025) (Airline and Retail): This benchmark evaluates user-agent interactions comprehensively within specialized real-world domains. We employ GPT-4.1 as the user simulator following the original experimental setting and report performance using the Avg@4 and Pass@4 metrics.

LLMs and Datasets. We source the training data from Ultra-FineWeb (Wang et al., 2025) and employ GLM-4.6 to generate 10K synthetic trajectories. These trajectories are then used to fine-tune both Qwen3-8B and Qwen3-32B. Additionally, we leverage the same 10K samples to train a data synthesizer model based on Qwen3-8B.

Baselines. We compare our synthetic data against the following open-source datasets: (1) **APIGEN-MT** (Prabhakar et al., 2025), (2) **Simia-Tau** (Li et al., 2025): We sample 50K multi-turn data from this dataset, (3) **MUA** (Zhao et al., 2025), and (4) **TOUCAN** (Xu et al., 2025): We sample 50K multi-turn data from this dataset. Notably, APIGEN-MT and Simia are in-domain training data generated in the τ -bench environment (Airline and Retail).

Training Details. All experiments are conducted on 2×8 NVIDIA H800 GPUs. For supervised fine-tuning, we set the learning rate to 5×10^{-6} and train for two epochs. Fine-tuning is performed using LLaMA-Factory (Zheng et al., 2024) under a full-parameter setting. Further details are provided in Appendix B.

4.2 Performance of GEM Synthesis Pipeline

BFCL V3 Results Table 1 presents performance comparisons on the BFCL V3 benchmark. The results of Simia are not included, as we observed its scores to be generally low, likely due to its reliance solely on in-domain τ -bench data. Our proposed GEM method demonstrates clear improvements over baseline models at both the 8B and 32B scales. In the 8B category, Qwen3-8B-GEM achieves an overall accuracy of 30.25%, significantly surpassing the base Qwen3-8B model and outperforming other open-source baselines such as APIGEN-MT and TOUCAN. The performance gain is even more pronounced in the 32B category, where Qwen3-32B-GEM attains an accuracy of 44.88%. This result not only exceeds open-source synthesized datasets like APIGEN-MT and MUA by a wide margin but also outperforms proprietary large-scale models, including GPT-4.1 (38.88%) and DeepSeek-V3.2-Exp (37.38%). These findings confirm that our data synthesis strategy effectively

Model	Multi-Turn Overall Acc	Multi-Turn Base	Multi-Turn Miss Func	Multi-Turn Miss Param	Multi-Turn Long Context
<i>Proprietary & Large Scale Models</i>					
GPT-4.1	38.88	47.50	32.50	32.50	43.00
DeepSeek-V3.2-Exp	37.38	41.50	39.50	33.50	35.00
Gemini-2.5-Flash	36.25	41.50	36.00	32.00	35.50
<i>8B Models</i>					
Qwen3-8B	18.00	24.00	17.00	13.50	17.50
APIGEN-MT	21.00	25.50	19.00	25.00	14.50
TOUCAN	21.88	27.00	21.50	21.00	18.00
MUA	21.13	29.50	18.00	20.00	17.00
Qwen3-8B-GEM	30.25	40.00	30.00	28.00	23.00
<i>32B Models</i>					
Qwen3-32B	28.35	34.00	24.50	25.50	29.50
APIGEN-MT	29.50	36.00	27.00	28.50	26.50
TOUCAN	35.00	41.00	37.50	26.00	35.50
MUA	26.25	30.00	25.00	23.00	27.00
Qwen3-32B-GEM	44.88	52.00	40.00	38.50	49.00

Table 1: **Performance comparison on the BFCL V3 benchmark.** The results are categorized by model scale. The best performance in each category is highlighted in **bold**. All metrics are reported as accuracy scores. The GEM-based models are highlighted in **blue**.

enhances function-calling capabilities across various multi-turn interaction categories.

τ^2 -Bench Results Table 2 presents the performance on the τ^2 -bench benchmark, covering the Airline and Retail domains. We would like to emphasize that our models are trained on synthetically generated data that is strictly *out-of-domain* with respect to the τ^2 -bench test sets, while APIGEN-MT and SIMIA are synthesized datasets within the τ environment (Retail and Airline), which can be regarded as in-domain training data. Despite this, our approach still demonstrates comparable performance. At the 8B scale, Qwen3-8B-GEM remains highly competitive, achieving results comparable to models like SIMIA that are fine-tuned on in-domain synthetic data, and surpassing APIGEN-MT in the Retail domain with a Pass@4 score of 75.44% versus 69.30%. At the 32B scale, our model exhibits strong generalization, outperforming both SIMIA and MUA in the Retail domain with a Pass@4 of 86.84% and delivering competitive results in the Airline domain. This indicates that our text-based synthesis pipeline instills a fundamental understanding of tool-use reasoning that transfers effectively to unseen, real-world domains, matching or even exceeding the performance of models trained on more domain-aligned data distri-

Model	Airline		Retail	
	Avg@4	Pass@4	Avg@4	Pass@4
<i>8B Models</i>				
Qwen3-8B	13.00	18.00	38.16	66.67
APIGEN-MT	23.50	42.00	42.54	69.30
Simia	35.50	52.00	43.20	70.18
TOUCAN	20.50	42.00	26.32	52.51
MUA	20.50	40.00	30.26	61.40
Qwen3-8B-GEM	22.00	40.00	44.52	75.44
<i>32B Models</i>				
Qwen3-32B	21.00	40.00	43.20	70.18
APIGEN-MT	36.00	52.00	44.52	74.56
Simia	38.00	62.00	48.03	73.68
TOUCAN	30.00	52.00	43.86	72.81
MUA	33.00	54.00	49.56	80.70
Qwen3-32B-GEM	35.50	56.00	55.48	86.84

Table 2: **Results on the τ^2 -bench (Airline, Retail).** We report avg@4 and pass@4 metrics. The GEM-based models are highlighted in **blue**. The best results within each model size are **bolded**.

butions. Overall, the results underscore the effectiveness of our proposed paradigm.

4.3 Performance of GEM Synthesizer

We use 10K trajectories generated by GEM synthesizer to fine-tune Qwen3-8B, as shown in Table 3. When using the same data source (Ultrafineweb),

our GEM-Synthesizer achieves an overall accuracy of 28.38% on the BFCL dataset, which is close to the performance obtained by directly using synthesized data generated by GLM-4.6, and also yields the Pass@4 score (73.68%) in the τ^2 Retail domain. We also use another textual source, Wikihow (Koupae and Wang, 2018), to show the generalization of our synthesizer. The *GEM-Synthesizer* also achieves a high overall performance on both BFCL (28.50%) and on the τ^2 -bench Airline domain. These findings indicate that our synthesizer can generate high-quality tool-use trajectories in a low-cost, end-to-end way, thereby enabling stronger generalization in complex tool-use scenarios.

4.4 Scaling Analysis

To investigate whether GEM can effectively enhance the tool-use capabilities of the model by scaling the volume of training data, we synthesize an additional 70K samples to evaluate the performance trend. As shown in Table 4, our approach continues to yield consistent and substantial improvements in the model’s tool-use capability as data scale increases.

4.5 Ablation Study

To validate the effectiveness of our synthetic data pipeline, we conduct an ablation study with BFCL V3 benchmark focusing on two key components: the refinement stage and the LLM-based check as shown in Table 5. We report the ablation results on τ^2 -bench in Appendix C.

Effect of Refinement. This stage substantially boosts performance. For instance, it enhances the performance of Qwen3-32B from 32.50% to 44.88%, an increase of over 12 percentage points. This process enhances the complexity and quality of synthetic trajectories, thereby enabling more effective learning of multi-turn tool use.

Effect of LLM-Based Check. This stage consistently improves results by filtering out samples with hallucinations or inconsistencies. For the 8B model, it raises overall accuracy from 27.38% to 30.25%. This mechanism also contributes to stronger performance across both model scales.

4.6 Data Analysis

Figure 5 presents a statistical overview of our synthesized multi-turn tool-use trajectories, highlighting three key dimensions: the number of distinct

tools employed, the number of messages per dialogue, and the total number of tool calls. On average, each trajectory involves 8.6 distinct tools, indicating that the synthesis process requires models to meaningfully select and combine multiple tools within a single task. Moreover, the trajectory contains an average of 46 turns. The considerable length of these dialogues ensures that tasks cannot be resolved simply, helping models learn to maintain context, track task state, and engage in multi-step planning over prolonged dialogues. In comparison, existing open-source datasets such as APIGEN-MT (Prabhakar et al., 2025) average around 18.5 turns, while TOUCAN (Xu et al., 2025) contains only about 6.24 turns. Finally, each trajectory averages 16.3 tool calls, while APIGEN-MT contains only an average of 4.3 tool calls. This high frequency underscores the multi-step, tool-driven nature of the synthesized tasks. Taken together, these statistics demonstrate that our synthesized dataset exhibits substantial diversity and interaction depth, making it well-suited for training models in multi-turn tool-use scenarios.

4.7 Case Study

For the case study illustrated in Figure 6, this synthesized trajectory is based on a real-world photo-editing scenario. The synthesis process first extract diverse tools from the text description, and identifies key constraints from the user’s actions, such as the restriction that font size must be between 8 and 96 points. Building on these tools and rules, the dialogue encompasses a variety of realistic interaction patterns: proactively clarifying missing parameters (i.e., image path and text placement), invoking tools in the correct sequence, rejecting requests that violate constraints (i.e., exceeding the font size limit) while offering compliant alternatives, and recovering from errors by retrying with another available printer. It guides the model to learn how to validate inputs, adhere to system constraints, execute tasks step by step, and respond flexibly to errors within multi-turn interactions, thereby achieving reliable tool-use capabilities. We also show a complete synthesized trajectory in Appendix E to help readers better understand our methodology.

5 Related Work

Tool-Use Data Synthesis To equip LLM-based agents with tool-calling capabilities, prior works focus on synthesizing tool-use training data. Tool-

Model	BFCL V3 Multi-Turn					Tau2			
	Overall	Base	Miss	Miss	Long	Airline		Retail	
	Acc	Acc	Func	Param	Context	Avg@4	Pass@4	Avg@4	Pass@4
Qwen3-8B (Base)	18.00	24.00	17.00	13.50	17.50	13.00	18.00	38.16	66.67
+ GEM-GLM (Ultrafineweb)	30.25	40.00	30.00	28.00	23.00	22.00	40.00	44.52	75.44
+ GEM-Synthesizer (Ultrafineweb)	28.38	41.50	23.50	27.50	21.00	26.00	40.00	42.11	73.68
+ GEM-Synthesizer (Wikihow)	28.50	43.50	22.00	27.00	21.50	25.00	42.00	39.67	68.42

Table 3: **Effects of data synthesizer and text sources.** We compare the impact of different trajectory generation models (GLM vs. Trained Synthesizer) and data sources (Ultrafineweb and Wikihow) on the Qwen3-8B base model. The best performance is highlighted in **bold**.

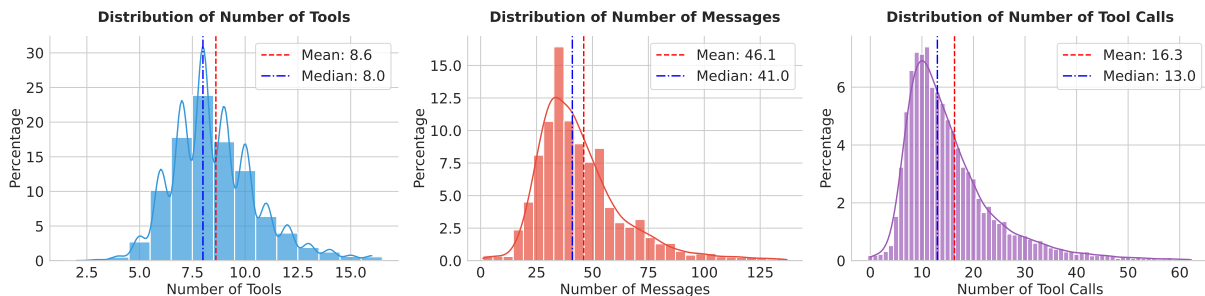


Figure 5: Data analysis. We find the synthesized trajectories are overall complex. **Left:** distribution of number of tools. **Mid:** distribution of number of messages. **Right:** distribution of number of tool calls in each trajectory.

Model	Airline		Retail	
	Avg@4	Pass@4	Avg@4	Pass@4
<i>8B Models</i>				
Qwen3-8B	13.00	18.00	38.16	66.67
GEM-8B (10K)	22.00	40.00	44.52	75.44
GEM-8B (70K)	29.50	56.00	52.19	79.82

Table 4: Performance scaling with respect to tool-use trajectory data volume.

Bench (Qin et al., 2023) builds a large-scale function-calling dataset over thousands of APIs. ToolACE (Liu et al., 2024a) designs an automatic agentic pipeline that iteratively expands an API pool and synthesizes verified function-calling traces with high complexity and diversity. For multi-turn interaction, APIGen-MT (Prabhakar et al., 2025) generates structured task blueprints with ground-truth action sequences and then simulates realistic human-agent dialogues grounded in executable APIs. MagNet (Yin et al., 2025) represents multi-tool workflows as graph-structured function paths and converts them into multi-turn conversations with executable calls. ToolACE-MT (Zeng et al., 2025) adopts a non-autoregressive framework that drafts entire dialogues in one shot and refines them via iterative editing and verifi-

Model	Overall Acc	Multi Base	Miss Func	Miss Param	Long Cxt.
Qwen3-8B (Base)	18.00	24.00	17.00	13.50	17.50
Qwen3-8B-GEM	30.25	40.00	30.00	28.00	23.00
(w/o Refine)	26.00	33.50	23.50	27.00	20.00
(w/o LLM-Based Check)	27.38	37.50	25.00	24.00	23.00
Qwen3-32B (Base)	28.35	44.00	24.50	25.50	29.50
Qwen3-32B-GEM	44.88	52.00	40.00	38.50	49.00
(w/o Refine)	32.50	40.00	30.00	27.00	33.00
(w/o LLM-Based Check)	44.25	52.00	40.50	40.00	44.50

Table 5: **Ablation study.** We conduct ablation study with BFCL V3 benchmark.

cation. TOUCAN (Xu et al., 2025) crawls MCP servers and synthesizing 1.5M of tool-use data. Unlike prior works that rely on pre-defined tools, our work introduces a novel paradigm that directly extracts multi-turn trajectories from text, thereby unlocking a authentic and scalable source of tool-use agentic data.

Tool-Use Capability Evaluation To systematically assess tool-use capabilities of LLM agents, a series of benchmarks have been proposed. ToolBench (Qin et al., 2023) focuses on evaluating whether models can translate natural-language instructions into correct API calls across thousands of real-world tools. The Berkeley Function Calling

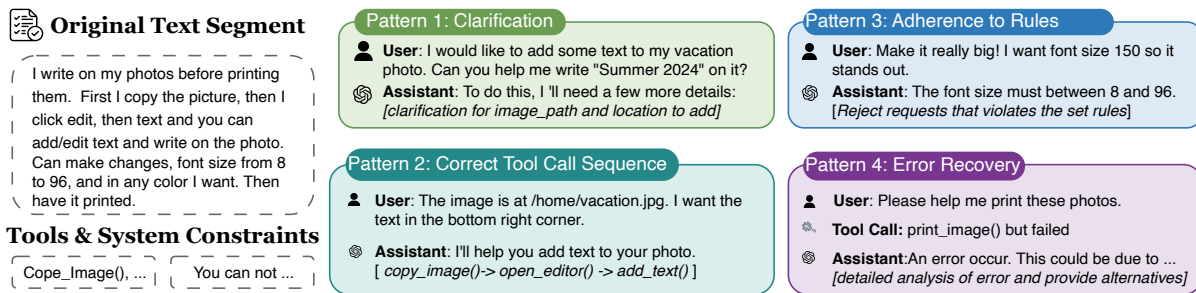


Figure 6: Case study of our generated multi-turn tool-use trajectory.

Leaderboard (BFCL) (Patil et al., 2025) provides a large-scale, syntax- and semantics-aware evaluation of function calling across diverse domains and programming languages, using AST-based checks to measure the accuracy and compositionality (e.g., parallel calls) of model-generated function invocations. ACEBench (Chen et al., 2025) evaluates multi-turn tool-use capability from three perspectives: normal, special, and agent. τ -bench (Yao et al., 2024) emulates dynamic conversations between a simulated user and a tool-augmented agent in domain-specific scenarios, jointly evaluating task success, tool selection, and adherence to domain policies. τ^2 -bench (Barres et al., 2025) further extends this line by introducing dual-control environments where both the agent and the user can invoke tools, highlighting the challenges of coordinating tool use and guiding user actions in more realistic customer-service-style tasks.

6 Conclusion

This paper presents a novel paradigm and the agentic data synthesis pipeline, which directly synthesize multi-turn tool-use trajectories from text corpora, effectively bypassing the dependency on predefined tools. Models trained on our data achieve significant performance gains on benchmarks, demonstrating the potential of leveraging open-world textual knowledge as a scalable source for advancing autonomous agents.

Limitation

While GEM offers notable advantages, several limitations highlight important directions for future work. First, our current framework is primarily designed for text-only large language models and focuses on English. Extending this approach to support multilingual contexts and multi-modal tools remains a valuable area for exploration. Second, given the vast scale of available textual corpora,

a promising avenue is to further harness these resources for mid-training, which could significantly enhance the scalability and effectiveness of our method.

Acknowledgments

We would like to thank all colleagues at Meituan for their support of this work.

References

- 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*.
- Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Yuefeng Huang, and 1 others. 2025. Acebench: Who wins the match point in tool usage? *arXiv preprint arXiv:2501.12851*.
- 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*.
- Ai Jian, Xiaoyun Zhang, Wanrou Du, Jingqing Ruan, Jiangbo Pei, Weipeng Zhang, Ke Zeng, and Xunliang Cai. 2026. Trust-sql: Tool-integrated multi-turn reinforcement learning for text-to-sql over unknown schemas. *arXiv preprint arXiv:2603.16448*.
- Mahnaz Koupaee and William Yang Wang. 2018. Wikihow: A large scale text summarization dataset. *arXiv preprint arXiv:1810.09305*.
- Jian Li, Yizhang Jin, Dongqi Liu, Hang Ding, Jiafu Wu, Dongsheng Chen, Yunhang Shen, Yulei Qin, Ying Tai, Chengjie Wang, and 1 others. 2026. Se-search: Self-evolving search agent via memory and dense reward. *arXiv preprint arXiv:2603.03293*.
- Yuetai Li, Huseyin A Inan, Xiang Yue, Wei-Ning Chen, Lukas Wutschitz, Janardhan Kulkarni, Radha Poovendran, Robert Sim, and Saravan Rajmohan. 2025. Simulating environments with reasoning models for agent training. *arXiv preprint arXiv:2511.01824*.

- Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, and 1 others. 2025. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.
- Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, and 1 others. 2024a. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*.
- Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh RN, and 1 others. 2024b. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *Advances in Neural Information Processing Systems*, 37:54463–54482.
- Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. 2025. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*.
- Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Awalganekar, 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*.
- Jinbo Su, Lingzhe Gao, Wei Li, Shihao Liu, Haojie Lei, Xinyi Wang, Yuanzhao Guo, Ke Wang, Daiting Shi, and Dawei Yin. 2025. Racqc: Advanced retrieval-augmented generation for chinese query correction. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 675–689.
- Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, and 1 others. 2025. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*.
- Yudong Wang, Zixuan Fu, Jie Cai, Peijun Tang, Hongya Lyu, Yewei Fang, Zhi Zheng, Jie Zhou, Guoyang Zeng, Chaojun Xiao, and 1 others. 2025. Ultrafineweb: Efficient data filtering and verification for high-quality llm training data. *arXiv preprint arXiv:2505.05427*.
- Zhangchen Xu, Adriana Meza Soria, Shawn Tan, Anurag Roy, Ashish Sunil Agrawal, Radha Poovendran, and Rameswar Panda. 2025. Toucan: Synthesizing 1.5 m tool-agentic data from real-world mcp environments. *arXiv preprint arXiv:2510.01179*.
- 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*.
- Zhiyuan Yao, Zishan Xu, Yifu Guo, Zhiguang Han, Cheng Yang, Shuo Zhang, Weinan Zhang, Xingshan Zeng, and Weiwen Liu. 2026. Toolace-mcp: Generalizing history-aware routing from mcp tools to the agent web. *arXiv preprint arXiv:2601.08276*.
- Fan Yin, Zifeng Wang, I-Hung Hsu, Jun Yan, Ke Jiang, Yanfei Chen, Jindong Gu, Long Le, Kai-Wei Chang, Chen-Yu Lee, and 1 others. 2025. Magnet: Multi-turn tool-use data synthesis and distillation via graph translation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32600–32616.
- Xingshan Zeng, Weiwen Liu, Lingzhi Wang, Liangyou Li, Fei Mi, Yasheng Wang, Lifeng Shang, Xin Jiang, and Qun Liu. 2025. Toolace-mt: Non-autoregressive generation for agentic multi-turn interaction. *arXiv preprint arXiv:2508.12685*.
- Xiaoyun Zhang, Jingqing Ruan, Xing Ma, Yawen Zhu, Jiansong Chen, Ke Zeng, and Xunliang Cai. 2025. A reasoner for real-world event detection: Scaling reinforcement learning via adaptive perplexity-aware sampling strategy. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 310–324.
- Weikang Zhao, Xili Wang, Chengdi Ma, Lingbin Kong, Zhaohua Yang, Mingxiang Tuo, Xiaowei Shi, Yitao Zhai, and Xunliang Cai. 2025. Mua-rl: Multi-turn user-interacting agent reinforcement learning for agentic tool use. *arXiv preprint arXiv:2508.18669*.
- Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyang Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. *arXiv preprint arXiv:2403.13372*.

A Prompts

A.1 Tag Annotation

Domain & Category Annotation

Determine whether the following text contains multi-step operations involving the use of an APP, website, computer, or other machine (such as robot, elevator, etc), if contains, generate one sentence summary of the task and identify the platform, domain and task category of the multi-step task.

Detailed Instruction

1. Platform category: operator, computer, phone, machine, other
2. Domain category: adult, arts_and_entertainment, autos_and_vehicles, beauty_and_fitness, books_and_literature, business_and_industrial, computers_and_electronics, finance, food_and_drink, games, health, hobbies_and_leisure, home_and_garden, internet_and_telecom, jobs_and_education, law_and_government, news, online_communities, people_and_society, pets_and_animals, real_estate, science, sensitive_subjects, shopping, sports, travel_and_transportation
3. Task category: databases, multimedia_processing, cloud_platforms, calendar_management, cryptocurrency, location_services, communication, search, file_systems, web_scraping, ecommerce_and_retail, customer_data_platforms, developer_tools, virtualization, version_control, research_and_data, aigc, travel_and_transportation, note_taking, language_translation, rag_systems, security_and_iam, social_media, monitoring, weather_services, customer_support, blockchain, knowledge_and_memory, financial_trading, marketing, enterprise_business_intelligence, transportation_logistics, iphone_android, smart_home, education_elearning, robot_control, website_control, gaming_entertainment

Input

[text]

Output Format

<multi_step>False</multi_step>

Or:

<multi_step>True</multi_step>

<summary>xxx</summary>

<domain>Shopping, Sports</domain>

<platform>Operator</platform>

<task>customer_support</task>

A.2 Workflow & Tool Discovery

Workflow & Tool Discovery

You are an program design expert. Given a workflow description in a scenario, your task is to design multiple functions to translate the execution process of this workflow into program.

Instruction

1. Extract all intermediate steps in the workflow, if the text contains multiple workflows, output them in a list.
2. Convert **every** step to a function and represent them as an execution graph (i.e. (login)->(search_query)->..)
3. Based on the execution graph, generate real API calls that populate the tools with reasonable parameters, simulating a use case of actual tool invocation steps.
4. Provide detailed API definitions used in the above process.
5. Follow the following steps to generate more complex workflows and tools:
 - Workflow Exploration: You need to explore multiple workflows or complex constraints that may exist in the document
 - These workflows represent the possible interaction patterns of a real user-agent.
 - Dependencies: "X must happen before Y".
 - Uniqueness/Limits: "Only one Admin allowed", "Name must be unique".
 - Conditionals: "If user is X, they cannot do Y".
 - Tool Design (Functional API Level)
 - Design a set of JSON-schema tools based on the text.
 - The required parameters of a tool need to be carefully considered and designed, mirroring the logic of the real world. For example, viewing system data typically requires authorization authentication, and providing user ID, product ID, etc.
 - It mimics a database structure and provides read and write tools. For example, it provides tools for querying user information, along with corresponding tools for modifying user information.
 - Each tool's name should be short and readable, semantically clear and general, reusable (e.g., "flight_search" rather than "flight_detailed_search_for_tom_2025")
 - Each tool should implement a single, coherent capability. It should not bundle multiple unrelated or multi-stage workflows into one tool. (e.g., create two tools "plan_trip" + "book_trip" rather than only one tool named "plan_and_book_trip")
 - Each tool's parameters should be

explicitly defined in the schema with clear types and meanings. Parameter names should be self-explanatory rather than cryptic (e.g., use "check_in_date" with type "string" and a short description, rather than a vague parameter named "d1").

- The majority of tools describe functional data operations that either retrieve information from or modify the state of the environment (e.g., get_status, update_permissions)

```
# Workflow Description
[text]
```

```
# Output Format
<workflow>
<description>short task description</description>
<steps>Step1: ... \n Step2: ... </steps>
<execution_graph>(api_name1)->(api_name2,
  api_name3)->...</execution_graph>
<actions>[{"name": "api_name", "arguments":
  {"arg_name": "value", ...}], ... (more
  API calls as required)]</actions>
<tools>[{"name": "api_name1", "description":
  "", "inputSchema": {"type": "object",
  "properties": {"arg_name1": {"type": "",
  "description": ""}, "arg_name2": {"type":
  "", "description": ""}}, "required": ["arg_name2"]}}, {"name": "api_name2",
  "description": "", "inputSchema": {"type":
  "object", "properties": {"arg_name1": {"type":
  "", "description": ""}, "arg_name2": {"type":
  "", "description": ""}}, "required": ["arg_name2"]}}]</tools>
</workflow>

<workflow>
(more workflows)
</workflow>
```

A.3 Trajectory Generation

Trajectory Generation

You are tasked with generating high-quality multi-turn dialogue trajectories based on a given text document. The trajectory should demonstrate an AI assistant helping users complete tasks while strictly following domain-specific rules and constraints.

You will be provided with:

- A list of Available Tool Candidates;
- A source text document that contains the description of the scenario and task steps;

Completion Requirements

1. System Prompt: Extract and explicitly state ALL important domain-specific rules and constraints from the source text document.

Example:
 <system>
 You are a agent specilized in retail domian.
 Here are some basic rules to follow:
 - An order can only be cancelled if its status is 'pending' ...
 - Modify action can only be called once, and will change the order status to 'pending (items modified)' ...
 - ...
 </system>

2. User Task: Create natural, progressive user requests that test the system's rule enforcement and constraint handling. Here are some features:
 - Naturalness: Requests should reflect real-world use cases
 - Ambiguity: User requests are often incomplete, requiring the assistant to analyze or clarify them.

Example: <user> I want to cancel order # W2575533. </user> (the user do not provide the specific reason, and the assistant should ask for clarification)
 ; <user> Recommend me a desktop. I often go out. </user> (the user do not explicitly state the attribute of the item, but the assistant should analyze and know it based on the stated preference)

- Consistent: User's intention, persona, and their behavior should be consistent across the dialogue.
- Complex: The user's request is challenging enough to test the assistant's ability. Users can make requests that violate domain rules and are not allowed to alert the assistant (e.g., do not ask the model to verify the order status first).

At least in one turn, the user's request is very complex and require assistant to handle it carefully.

Example 1:

<user> I need to make several changes to my order #W2575533. Can I change the E-Reader to a different size, swap the Garden Hose color, and also update my shipping address </user>

(Requires assistant to: check order status, verify each item can be modified, handle address change separately, remind about one-time modification limit)

Example 2:

<user> Check my tire pressures. If any of them are low, find me the nearest service station and also check if I have enough fuel to get there

(Requires: check tire pressure, evaluate condition, conditionally call find_nearest_shop, check fuel level, calculate if sufficient)

Example 3:

<user> I'm planning a three-day trip starting from Hangzhou, and I need help creating an itinerary. One more thing

about the second day - I'm trying to be smart about my budget. If I end up booking a luxury hotel that costs 800 CNY or more per night, then I need to be more careful with other expenses: my total spending on both restaurants (lunch and dinner) should stay under 350 CNY, both restaurants should be rated at least 4.0 stars, and the afternoon attraction ticket needs to be less than 120 CNY. </user>

(Requires: check multiple constraints)

3. Assistant: Generate Responses that Demonstrate Rule Enforcement, Clear Communication, and Intelligent Problem-Solving:

- Reasoning and Adaptive Planning: The Assistant should reason through problem contexts and plan appropriate steps. Sometimes users may not be able to directly provide the parameters for tool calls, and the assistant needs to accurately consider whether the parameter values can be obtained through other information and tools.
- Precondition Checks: Before executing tasks, the Assistant should validate any necessary preconditions (e.g., authenticating identity, verifying the status of an order)
- Domain Rules and Constraints: The Assistant must follow domain-specific rules at all times. Ensure the assistant's tool call and response genuinely addresses those requirements.
- User-Centric Principle: The Assistant must accurately understand and satisfy all user needs and preferences without breaking domain rules. For example, if user states "prefers A, wants B, and tell me C", the assistant should satisfy all requirements.
- No Hallucination:
 - Context Faithfulness: Maintain absolute fidelity to tool outputs. If data contradicts user expectations, explicitly report the discrepancy instead of distorting facts to force a match.
 - Tool call arguments: The Assistant must only use argument values that are explicitly provided or implied by the user. It must not fabricate IDs, names, or other parameters; if any required value is missing or unclear, the Assistant should ask the user to supply it before calling the tool.
- Consequence-aware: Before executing any write operation that modifies the environment, the Assistant must actively think and assess its impact. For changes that are irreversible, the assistant should obtain explicit user confirmation before proceeding.
- Limitations: If the current request is beyond the assistant's ability, the Assistant must communicate this

limitation clearly. For example: If the user's needs are beyond domain rules, the Assistant must explain the limitation but should still respect the user's needs. (e.g., "I cannot do A, but I can do B. Should I proceed with B ?" -> Wait for confirmation).

- Correct Tool Calls:

- Tool call format: `<func>{"name": "exact_tool_name", "arguments": {"arg": "value"}}</func>`.
- Exact Matching: The Assistant must ensure that tool calls exactly match the available candidate tools. Calling tools that do not exist is prohibited.
- Parameter Validation: The arguments passed to the tools must exactly match the tool definitions, including the required parameters. The Assistant should avoid any missing parameters and validate that the values are accurate.

4. Tool Responses: Structured and Complete

- Correctness of Tool Call: A success response should only be returned if the tool call is correct (including both the correct function name and parameters). If any part of the tool call is incorrect or incomplete, the tool should return a failure response, indicating what went wrong
- Success Response: The tool's success response must return all relevant information in a well-structured format (e.g., JSON). This includes not just the requested data, but also any other relevant details, such as order ID, status, product ID, item ID, etc., in case the user requires further context.
- Error Response: When there is an error, the tool should return only the error message. No additional information should be provided that directly aids the Assistant in recovering from the error.
- Consistent Response Structure: The format and content of the tool's responses should remain consistent throughout the conversation. This ensures clarity and reliability in the tool's output, helping to maintain a smooth user experience.
- Don't confuse the order between turns:
 - a user message or tool result should be followed by an assistant message. (`<user>...<user> / <tool>...</tool> -> <assistant> ... </assistant>`)
 - If the assistant message includes a tool call, it can be followed by a tool message; (`<assistant>...<func>...</func></assistant> -> <tool>...</tool>`)
 - otherwise, it should be followed by a user message. (`<assistant> ... </assistant> -> <user> ... </user>`)
 - Tool result cannot followed by user message. (MUST NOT OUTPUT: `<tool>...</tool> -> <user>...</user>` (incorrect example))

5. Trajectory Pattern: The trajectory should exhibit multiple interaction patterns, with at least 3 distinct patterns appearing in total. Each individual pattern should be used at most 2 times within a single trajectory

[Pattern 1: Domain Rules & User Need conflicts]

[Pattern 2: Error Recovery]

[Pattern 3: Clarification and Disambiguation]

[Pattern 4: Assistant's Multi-hop Reasoning]

[Pattern 5: Assistant's Awareness of Domain Rules]

Other Requirements

- Always use English.
- The whole trajectory should be reasonable, realistic (conform to real-world dialogue scenarios and interactions), and fit the context of multi-turn tool usage.
- Don't confuse the order between rounds: a user message or tool result should be followed by an assistant message. If the assistant message includes a tool call, it can be followed by a tool message; otherwise, it should be followed by a user message. Tool result cannot be followed by user message.
- Tool call format:

```
<func>
{{"name": "exact_tool_name_in_toolsets", "
arguments": {"arg": "value"}}}}
</func>
```

Given Inputs

Available Tool Candidates
{candidate_tools}

A source text document consisting of
Task Steps Description
{current_task}

Output Format

You must STRICTLY follow the following output format.

Ensure ALL tags are properly opened and closed. Conversations like "<tool> ... </assistant>", "<assistant>...</user>" are wrong!!!

```
<system>
...
</system>
<user>
...
</user>
<assistant>
...
<func>
{{"name": "...", "arguments": {...}}}
</func>
</assistant>
```

```
<tool>
...
</tool>
```

A.4 Trajectory Refinement

Trajectory Refinement

You are rewriting a complex, realistic multi-turn tool-use trajectory for agentic training.

Goal: The trajectory should be complex, natural, no-hallucination and show the wisdom of the assistant.

It must show the assistant's ability for correct tool use, reasoning, context understanding, and communication skills with users.

Please carefully consider what problems exist with the existing synthetic trajectories and how to improve them to create high-quality trajectories.

You can follow the following guidelines.

Guideline

System Prompt Complexity

You need to refine and upgrade the constraints of the system prompts to make them more structured, systematic, and consistent with real-world logic, in order to fully test the agent's ability to make correct tool calls in complex scenarios. You also need to define the database schema in the system prompts, and the data structure returned by the tool will be based on this.

User Request Complexity & Naturalness

- Natural: Natural requests may include colloquial language, implied context, vague references to prior steps, or real-world motivations (e.g., saving time/money, lose weight). Avoid overly formal or purely instructional language
- User diversity: Create a user profile and maintain the user's personality and characteristics throughout the conversation history.
- Requires Deep Analysis & Reasoning: The user's request must necessitate careful analysis and multi-step reasoning to identify the correct tool(s) and determine appropriate parameter values.
- Requires Analysis of Tool Dependencies & Outputs: The request should force the assistant to understand dependencies between tools and use outputs from previous steps to decide which tool to invoke next.
- In at least 1 turn, the user's request contains multiple constraints, including explicit constraints, implicit requirements that require the

- assistant to infer.
- In at least 1 turn, The user asks a question that can only be answered by reasoning across outputs from multiple tool calls in the long context.
- Challenging Pitfalls: ****MUST INCLUDE AT LEAST 1-2 PITTFALLS**** for one trajectory.

This pattern sets traps to test the assistant's ability to correctly make robust tool calls based on rules, constraints or user preferences.

The trajectory must include request for this kind of challenging tool calls, and the assistant must explicitly analyze and identify these pitfalls to achieve robust tool calls. For example:

- Example 1

Rule: Each order can only be processed for return once.

User: "Please help me return both items from this order."

Assistant:

Correct: Execute `return_items([item1, item2])` in one call.

- Example 2

User: "Fill the fuel tank until we are able to reach Rivermist. Oil costs money so I just need to reach there."

Correct: `displayCarStatus("fuel")`, `fillFuelTank(fuelAmount=44)`

Wrong: `fillFuelTank(fuelAmount=50)`

Reason: The user wants to save money; filling the tank fully goes against their stated constraint.

- Example 3

Context: The initial working directory is alex.

User: "Go into the directory named after me and list all the visible and hidden contents in the current directory now, please."

Wrong: `cd(folder='alex')`, `ls()`, `mkdir(dir_name='alex')`

Reason: The model did not realize it was already in the alex directory and proceeded to modify the state by creating a new directory. The model should explore the current workdir first, thereby regarded as implicit tool call requirement.

Assistant Intelligibility

Demonstrate the following capabilities of the assistant:

(1) Communication Skills

- User Intent Understanding: Accurately interpret the underlying goals and context of the user's request.
- Confirmation & Clarification: Proactively confirm key details or ask the user to clarify ambiguous information when necessary.
- Capability Limitation Awareness: Clearly communicate the boundaries and

limitations of the assistant's available capabilities.

- Result Explanation & Summarization: Interpret, distill, and present tool outputs or complex information in a structured, understandable manner.
- Proactive Assistance: Anticipate potential user needs based on context and offer helpful suggestions in advance.

(2) Robust Tool-Calling Capability

- Tool Selection: Choose the most appropriate tool from the available set based on task requirements.
- Sequential Tool Usage: Plan and execute multi-tool workflows with correct dependencies and order.
- Parameter Handling: Correctly construct and validate complex argument structures (e.g., nested objects, lists), respecting type, range, and format constraints.
- Result Analysis: Parse and evaluate tool responses, extract relevant information, and determine validity for subsequent steps.
- State Tracking: Maintain awareness of completed and pending steps in a multi-step task.
- Constraint Analysis: Identify and adhere to real-world constraints and business rules (e.g., date ranges, mutually exclusive fields, batch limits).
- Error Handling: Gracefully manage tool failures, diagnose error causes, and adjust strategy or inform the user appropriately.
- Context Management: Effectively retain and utilize conversation history to ensure coherence across complex interactions.

(3) Reasoning & Execution Ability

- Planning & Task Decomposition: Break down complex or vague user requests into clear, executable step-by-step plans.
- Prerequisite Management: Recognize and acquire necessary information or conditions before executing tasks (e.g., querying environment, requesting user input).
- Verification and Validation: Perform essential checks before critical operations, such as exploring available tools, confirming permissions, or validating inputs.

Realistic & Complex Environment

IMPORTANT NOTES:

- The tool called in the trajectory MUST exist in candidate toolsets, with the parameter type and value correct.
- If it's necessary to add tools when constructing complex trajectories, you can add them in the final output tools block.

GOALS:

- This increases the difficulty of choosing tools. A diverse range of tools must be included, specifically reading and writing tools.
- Increase the difficulty of making totally correct tool calls (parameter type, value). For example, include structured inputs (list, dict, nested objects) and meaningful constraints.
- The tools should conform to the database schema as much as possible (may defined in system prompt), and mainly include read-write tools. You can add & modify tools.
- Unique: Tool call parameters should use unique database fields (such as user ID , product ID, etc.) as much as possible to mimic real logic.
- Realistic: Always avoid using empty placeholders. For example, do not return something like "5+ more results ...", "path=/example".
- Success Response: A success response should only be returned if the tool call is correct. The tool's response must return a complete data structure in a well-structured format (e.g., JSON).
- Error Response: You are allowed to simulate non-simple errors that might occur in the real world. The tool's response must return a concise error message. Avoid directly telling the assistant how to solve the problem.

Trajectory Diversity

- Reduce the frequency of repeatedly using certain tools to solve problems, avoiding the reduction of trajectory diversity, and retain only the most valuable trajectories for learning.
- Include but not limited to the following pattern, and make the following pattern more difficult, diverse and natural.
 - [Pattern 1: Environment Complexity]
 - [1.1: Error Recovery] In multi-turn function calling, models may encounter errors, such as invalid input or failed execution that require recovery. If you think of any suitable, non-trivial, real-world scenario errors, please include this pattern.
 - [1.2: Long Context] Introduce large volumes of extraneous data to test how well the model can extract crucial details from an overwhelming array of information.
 - [Pattern 2: Clarification] Tests the model's ability to recognize when essential information is missing from the user request.
 - [2.1: Can be inferred from the system] The assistant actively explore ways to find the essential information and complete the task
 - [2.2: Can not be inferred from the system] The assistant actively clarifies the situation.

- [Pattern 3: Identify Limitation] Requires the model to identify that no available function can fulfill the user request.
- [Pattern 4: Assistant guides user operation] For example, if a user reports no internet access, the assistant uses tool calls to discover that the SIM card is not inserted, and then guides the user to insert it (this process cannot be performed by the assistant alone due to real-world physical constraints and requires active communication and guidance between the assistant and the user).

```
# Input Data
Toolset: {tools}
Trajectory: {our_traj}
```

Output Requirement

- Reduce redundancy: Reduce the frequency of repeatedly using certain tools to solve problems, avoiding the reduction of trajectory diversity, and retain only the most valuable trajectories for learning.
- Preserve turn order.
 - The assistant can only call the tool once per round.
 - Each tool message must be followed by an assistant message.
 - If an assistant message contains a tool call, it must be followed by a tool message (tool result).
 - Tool messages must not be followed directly by user messages.
- If the original trajectory violates these rules or misuses tools, fix it in the rewritten version.
- Output all candidate tools (the original tools + new tools if needed)
- You must strictly follow the following output format.

Output format

```
<toolsets>
All candidate tools in JSON, OPENAI format.
[
  {{
    "name": "",
    "description": "",
    "inputSchema": {{
      "type": "",
      "properties": {{ }},
      "required": []
    }}
  }},
  ...
]
</toolsets>

<system>
[role and domain rules here]
</system>
<user>
...
</user>
```

```

<assistant>
...
<func>
{"name": "...", "arguments": {...}}
</func>
</assistant>
<tool>
[concrete tool response in JSON format if
  tool calls are made]
</tool>
<assistant>
...
</assistant>
...
(more conversations here)
"""

```

A.5 Hallucination Detection

Hallucination Detection Prompt

You are given a multi-turn tool-use trajectory. Please evaluate the trajectory according to the following rubrics.

Your job is to score the trajectory on the following binary rubrics.

For EACH rubric, you must output 0 or 1 only, according to the criteria below.

For each rubric, if there is no hallucination of the following content throughout the entire trajectory, the corresponding rubric score is 1.

If any single round does not meet (the condition), the corresponding rubric should be scored as 0. Be strict in your evaluation.

Rubric

R1: Tool-call hallucination

- Check whether any tool call uses argument values that are not provided or reasonably derivable from the dialogue context.

R2: Capability hallucination

Check whether the assistant makes incorrect claims about what can or cannot be done.

- H2-a False inability: The user request IS solvable using the available tools, but the assistant claims it cannot be done or refuses without justification.
- H2-b Missing limitation disclosure: The user request is NOT solvable with the available tools, but the assistant proceeds as if it is solvable, or fails to clearly explain the limitation and offer the closest feasible alternative.

R3: Context hallucination

Check whether the assistant misinterprets the ongoing context or references things that are not true in the dialogue.

- Wrongly referencing previous user constraints, preferences, or decisions.
- Cross-turn inconsistency: changing entities/values (IDs, counts, dates, constraints) without new evidence or tool output.
- Conflicting summaries: later summary contradicts earlier established facts.

Input
{trajectory}

Output Format
Return a single JSON object with EXACTLY these keys and integer values 0 or 1:

```

{{
  "R1": 0 or 1,
  "R2": 0 or 1,
  "R3": 0 or 1,
}}

```

Do NOT output anything else (no explanations, no comments).

B Hyper-parameter Setting

We use the sample hyper-parameter during the experiments. Please refer to Table 6 for details.

Hyperparams	Values	Hyperparams	Values
learning rate	5e-6	weight decay	0.05
warmup ratio	0.1	max length	32K
lr scheduler	cosine	batch size	64
epoch	2	BF16	True
Deepspeed	zero3	tool-call template	Hermes

Table 6: SFT Hyperparameters used.

C Ablation Study on τ^2 -bench

We report the ablation results of τ^2 -bench (Airline, Retail) in Table 7.

Model	Airline		Retail	
	Avg@4	Pass@4	Avg@4	Pass@4
Qwen3-8B	13.00	18.00	38.16	66.67
Qwen3-8B-GEM	22.00	40.00	44.52	75.44
w/o refine	25.00	36.00	41.23	74.56
w/o LLM-Based Check	22.00	36.00	42.76	71.05
Qwen3-32B	21.00	40.00	43.20	70.18
Qwen3-32B-GEM	35.50	56.00	55.48	86.84
w/o refine	31.00	56.00	40.35	73.68
w/o LLM-Based Check	35.00	52.00	56.80	82.46

Table 7: Ablation Study on τ^2 -bench (Airline, Retail).

D Domain Analysis

We analyze the domain distribution in Figure 7. The results demonstrate the diver domains existing in the raw text corpora.

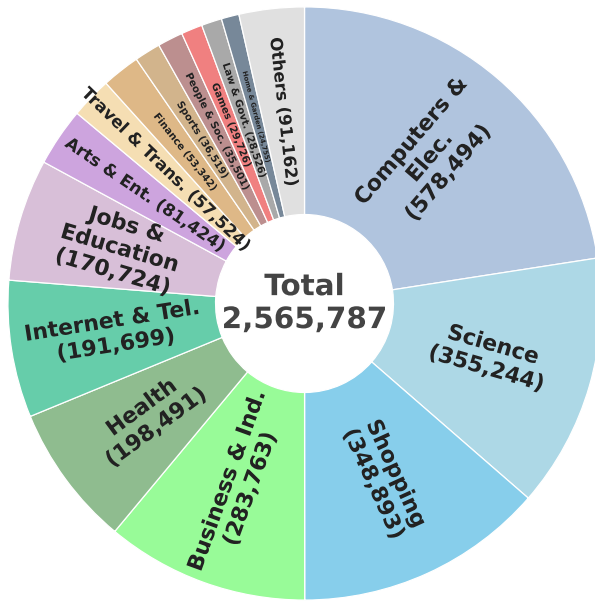


Figure 7: Domain Distribution.

E Example of Synthesized Trajectory

Please refer to Figure 8 and 9. The original text provides unstructured information regarding photo frame products, measurement methods, and specific size constraints. The system prompt establishes the persona of a custom framing specialist, enforcing a strict business workflow from user authentication to final order placement. A set of functional tools was designed to support this, including authentication, dimension calculation, constraint checking, and order processing. The final trajectory demonstrates the assistant's exceptional tool-calling and logical reasoning capabilities, particularly its ability to adapt by adjusting parameters and replanning the tool chain after an initial constraint violation, ultimately ensuring a successful transaction.

F Use Of AI Assistant

We mainly use large language models to polish our paper.

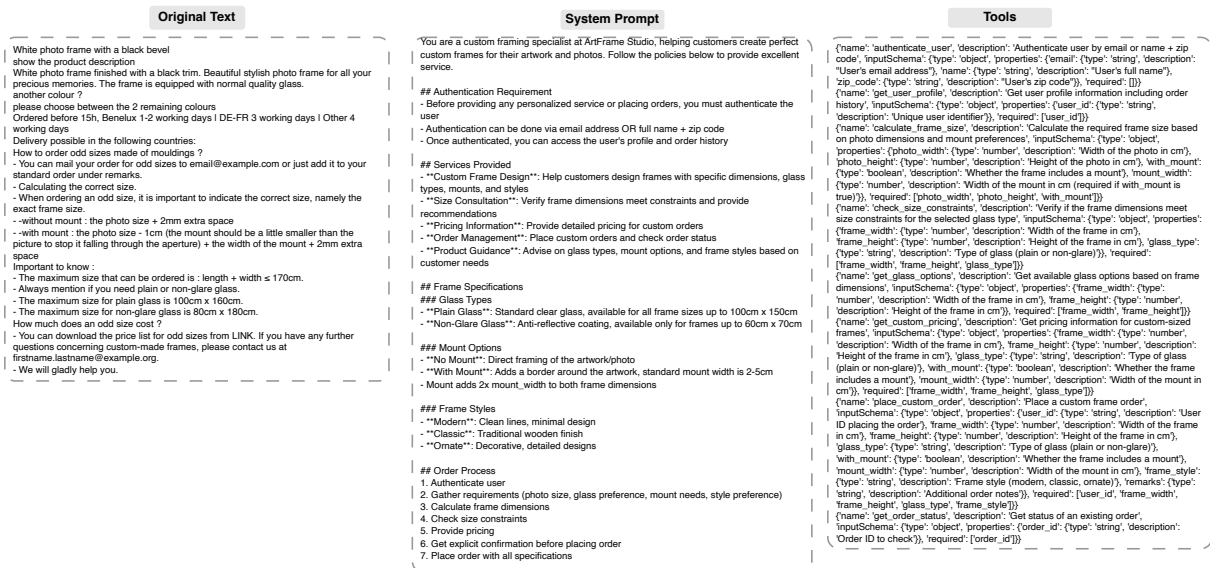


Figure 8: Full example of our generated trajectory (Part 1).

Full Synthesized Trajectory

user
Hi there! I need help ordering a custom frame for a large photograph I took. It's 50cm by 70cm, and I want to make sure it looks professional for an exhibition.

assistant
Hello! I'd be happy to help you create a professional custom frame for your exhibition photograph. A 50x70cm photo is quite impressive! Before we proceed with designing your frame, I'll need to authenticate your account to provide personalized service. Could you please provide either:
- Your email address, OR
- Your full name and zip code
Once I have that information, I can help you select the perfect frame options for your exhibition piece. We'll consider factors like glass type, mount options, and frame style to ensure your photograph looks its best. What's your email address or name and zip code?

user
My email is sarah.jenkins@email.com

assistant
<tool_call>
{\"name\": \"authenticate_user\", \"arguments\": {\"email\": \"sarah.jenkins@email.com\"}}
</tool_call>

user
<tool_response>
{\"user_id\": \"USR_789456\", \"status\": \"success\", \"message\": \"User authenticated successfully\"}
</tool_response>

assistant
Perfect! I've found your account, Sarah. Now let's design the perfect frame for your 50x70cm exhibition photograph.

To create the best presentation for your piece, I need to understand your preferences:

- \"Glass Type\"**: Would you prefer plain glass or non-glare glass? (Non-glare is great for exhibitions as it reduces reflections, but has some size limitations)
- \"Mount\"**: Would you like a mount (border) around your photo? This can add a professional touch and create visual breathing room. If yes, what width would you prefer? (2-5cm is standard)
- \"Frame Style\"**: We offer modern, classic, or ornate styles. Which would best complement your photograph and exhibition aesthetic?
- \"Budget\"**: Do you have a budget range in mind? This will help me recommend the best options within your price range.

Let's start with the glass type - what's your preference?

user
I definitely want non-glare glass since it's for an exhibition. And yes, I'd like a mount - maybe 3cm would be good. For the style, I'm thinking modern to keep it clean. My budget is around \$200-250.

assistant
Great choices! Non-glare glass will definitely help with exhibition lighting, and a 3cm mount with a modern frame will give it that professional, clean look. Let me calculate the frame dimensions with your mount preference and check if everything works with the non-glare glass option.

<tool_call>
{\"name\": \"calculate_frame_size\", \"arguments\": {\"photo_width\": 50, \"photo_height\": 70, \"with_mount\": true, \"mount_width\": 3}}
</tool_call>

user
<tool_response>
{\"frame_width\": 56, \"frame_height\": 76, \"message\": \"Frame dimensions calculated with 3cm mount on all sides\"}
</tool_response>

assistant
Perfect! With a 3cm mount, your frame will be 56cm x 76cm. Now let me check if these dimensions work with non-glare glass, as it has specific size constraints.

<tool_call>
{\"name\": \"check_size_constraints\", \"arguments\": {\"frame_width\": 56, \"frame_height\": 76, \"glass_type\": \"non-glare\"}}
</tool_call>

user
<tool_response>
{\"status\": \"error\", \"message\": \"Frame dimensions 56x76cm exceed non-glare glass maximum size.\"}
</tool_response>

assistant
I need to let you know about an important constraint: Your frame dimensions of 56x76cm exceed the maximum size for non-glare glass, which is limited to 60x70cm. The height of 76cm is within limits, but we need to address this. Here are your options:

- \"Reduce mount width\"**: If we use a 2cm mount instead, the frame would be 54x74cm, which would work with non-glare glass.
- \"Use plain glass\"**: This would allow your preferred 3cm mount (56x76cm frame) and is actually less expensive.
- \"Consider a different approach\"**: We could explore other creative solutions. Given your exhibition needs and budget of \$200-250, which option would you prefer? The 2cm mount with non-glare glass would still look very professional and keep you within budget.

user
Hmm, that's disappointing. I really wanted the non-glare glass. Let's go with the 2cm mount then. Will that still look good for an exhibition?

assistant
Absolutely! A 2cm mount will still look very professional for an exhibition - it's actually a common choice that provides just enough breathing room without overwhelming the artwork. Many galleries use this exact specification. Let me recalculate with the 2cm mount and get the pricing for you.

<tool_call>
{\"name\": \"calculate_frame_size\", \"arguments\": {\"photo_width\": 50, \"photo_height\": 70, \"with_mount\": true, \"mount_width\": 2}}
</tool_call>

user
<tool_response>
{\"frame_width\": 54, \"frame_height\": 74, \"message\": \"Frame dimensions calculated with 2cm mount on all sides\"}
</tool_response>

assistant
Perfect! Now your frame will be 54cm x 74cm with the 2cm mount. Let me verify this works with non-glare glass and get you the pricing.

<tool_call>
{\"name\": \"check_size_constraints\", \"arguments\": {\"frame_width\": 54, \"frame_height\": 74, \"glass_type\": \"non-glare\"}}
</tool_call>

user
<tool_response>
{\"status\": \"success\", \"message\": \"Frame dimensions 54x74cm are within non-glare glass constraints\"}
</tool_response>

assistant
Excellent! The 54x74cm dimensions work perfectly with non-glare glass. Now let me get the pricing for your custom frame with all your specifications.

<tool_call>
{\"name\": \"get_custom_pricing\", \"arguments\": {\"frame_width\": 54, \"frame_height\": 74, \"glass_type\": \"non-glare\", \"with_mount\": true, \"mount_width\": 2}}
</tool_call>

user
<tool_response>
{\"base_price\": 185, \"glass_upgrade\": 25, \"mount_cost\": 15, \"total\": 225, \"currency\": \"USD\", \"breakdown\": {\"frame\": 185, \"non_glare_glass\": 25, \"mount\": 15}}
</tool_response>

assistant
Great news! Your custom frame will cost \$225 total, which is perfectly within your \$200-250 budget. Here's the breakdown:
- Modern frame (54x74cm): \$185
- Non-glare glass upgrade: \$25
- 2cm mount: \$15
- **Total: \$225****

This includes professional assembly and will be exhibition-ready. The non-glare glass will significantly reduce reflections, and the modern style with the 2cm mount will give your photograph that clean, professional gallery look. Would you like me to place this order for you? I'll need:
- Confirmation to proceed with the order
- Any special delivery instructions or deadline for your exhibition

Should I go ahead and place this custom frame order?

user
Yes, please place the order! The exhibition is in 3 weeks, so I need it by then. Can you make sure it's delivered in time?

assistant
<tool_call>
{\"name\": \"place_custom_order\", \"arguments\": {\"user_id\": \"USR_789456\", \"frame_width\": 54, \"frame_height\": 74, \"glass_type\": \"non-glare\", \"with_mount\": true, \"mount_width\": 2, \"frame_style\": \"modern\", \"remarks\": \"Exhibition deadline - needed within 3 weeks\"}}
</tool_call>

user
<tool_response>
{\"order_id\": \"ORD_20240315_789\", \"status\": \"confirmed\", \"estimated_delivery\": \"2024-03-25\", \"message\": \"Order placed successfully. Express processing selected for exhibition deadline.\"}
</tool_response>

assistant
Perfect! Your custom frame order has been placed successfully. Here are your order details:
\"Order Confirmation: ORD_20240315_789**
- Frame: 54cm x 74cm modern style
- Glass: Non-glare
- Mount: 2cm white mount
- Total: \$225
- ****Estimated Delivery: March 25, 2024**** (within your 3-week deadline!)

I've noted your exhibition deadline in the order, and express processing has been automatically selected to ensure timely delivery. You'll receive a confirmation email shortly with tracking information once your frame ships. Is there anything else you'd like to know about your order?

Figure 9: Full example of our generated trajectory (Part 2).