RETHINKING DATA GENERATION FOR LONG-HORIZON TOOL CALLING

Anonymous authors
Paper under double-blind review

ABSTRACT

Real-world tasks routinely require long sequences of tightly interdependent tool calls, yet existing benchmarks for evaluating agentic systems rarely reflect this complexity. While current benchmarks capture realistic multi-turn interactions and underspecified inputs, they focus on shallow execution trajectories with limited inter-function dependencies. We examine current data generation approaches for tool-calling benchmarks and argue that they inherently bias datasets toward low-complexity examples. We propose BRIT (**B**ack translating and **R**efining **I**teratively for **T**rajectory generation), which makes two systematic modifications compatible with existing pipelines: (1) generating trajectories before queries to optimize complexity explicitly, and (2) applying iterative refinement instead of filtering to preserve complex instances. We provide theoretical analysis showing two complementary results. First, filtering during post-processing inevitably produces a light-tailed complexity distribution, regardless of the initial complexity distribution. Second, when the generator already produces a heavy-tailed complexity distribution, per-node refinement budgets that grow logarithmically with trajectory complexity are sufficient to preserve that heavy tail after post-processing. Empirically, BRIT substantially increases the complexity of generated datapoints. Our work advocates for including high-complexity instances in tool-calling evaluation and provides practical, scalable methods for achieving this goal.

1 INTRODUCTION

Agentic applications that interact with multiple tools to fulfill user requests have become a central use case for modern language models (Qin et al., 2023; Patil et al., 2024; Mu et al., 2023; Liu et al., 2024b; Ye et al., 2024). Interfacing with external APIs and toolkits not only broadens the scope of what these systems can accomplish (Qin et al., 2023; Ye et al., 2024; Zeng et al., 2023), but also helps mitigate hallucinations by enabling retrieval from reliable information sources (Schick et al., 2023; Gao et al., 2023; Zhao et al., 2024).

Given the growing adoption of tool-calling applications, it is essential to rigorously evaluate the tool-calling capabilities of large language models (LLMs) to ensure reliable deployment. Accordingly, researchers have developed a range of benchmarks to systematically assess different facets of tool use, including single-turn and multi-turn function calling (Qin et al., 2023; Zhuang et al., 2023; Yan et al., 2024; Guo et al., 2024; Wang et al., 2024), interaction within dynamic agent environments (Yao et al., 2024; Chen et al., 2025), and the handling of underspecified (Zhang et al., 2024; Yang et al., 2024) or potentially harmful requests (Andriushchenko et al., 2024; Luo et al., 2025). However, existing evaluations largely overlook *compositional queries that require long sequences of interdependent function calls*.

A closer examination of existing benchmark construction practices reveals that data curation is largely driven by the use of LLMs as synthetic data generators (Liu et al., 2024a; Qin et al., 2023; Liu et al., 2024b; Charlie Cheng-Jie Ji, 2024). In this paper, we argue that the implementation of these data generation pipelines inherently constrains the production of high-complexity datapoints, where complexity is characterized by the length of execution traces / the number of different function calls they entail. More specifically, the following two design choices limit the complexity of data generated:

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Choice 1: Direct Query Synthesis. Given a tool set and an intent or problem type, most approaches use LLMs to sequentially generate a query and then its solution. This sequentiality makes it difficult to control query complexity, since complexity is inferred only through the resulting solution trajectory (through metrics such as **execution length** and **degree of data dependencies**) rather than estimated *a priori*. Since trajectory construction follows a fixed query, the difficulty of the query directly amplifies the likelihood of errors — creating a structural bias where only easier instances tend to survive as valid datapoints.

Choice 2: Filtering as Post-processing. Because synthetic data generation is inherently error-prone, most pipelines (Qin et al., 2023; Patil et al., 2024; Chen et al., 2025; Yin et al., 2025) apply post-processing steps to ensure that only valid examples are included in the final dataset, often by using an LLM as a judge (Li et al., 2023; Tang et al., 2023; Liu et al., 2024a;b) to filter out erroneous instances. However, since simpler datapoints are more likely to be judged valid than more challenging ones, this practice disproportionately retains low-complexity examples, thereby increasing their concentration and further exacerbating the issue. We also provide theoretical justification by adapting the error model from Dziri et al. (2023) and demonstrating that *the density of valid trajectories exhibits a light-tailed distribution regardless of the underlying complexity distribution of trajectories generated before post-processing.*

These observations motivate two modifications in **BRIT**. (a) We adopt a **back-translation strategy** in which the function-call trajectory is generated first, followed by the corresponding query. This design allows complexity to be optimized explicitly at the trajectory level, making it easier to synthesize highly complex datapoints. Moreover, generating the query after the trajectory makes it easier to produce a query that is consistent with the underlying trajectory. (b) Instead of filtering datapoints, we apply **LLM-based refinement as a post-processing step**, increasing the likelihood that harder instances pass validation and are included in the final dataset. Refer fig. 1 to see how these modifications can improve the complexity distribution of the generated trajectories.

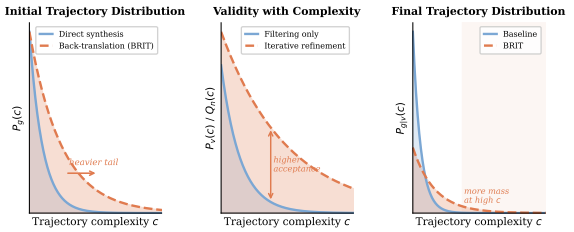

Figure 1: **Conceptual illustration of how BRIT improves the final trajectory distribution.** Back-translation shifts the initial generator distribution toward higher complexity (left), while iterative refinement raises the acceptance probability for complex trajectories (middle). Together, these yield a final valid distribution with substantially more mass at high complexity (right).

Using these modifications, BRIT achieves a higher-complexity trajectory distribution more efficiently than prior baselines — without requiring repeated resampling to obtain a sufficient number of complex valid instances. Beyond this, we show that *a logarithmically growing refinement budget provably recovers the heavy tail of the initial complexity distribution* — meaning that iterative refinement not only improves data quality in practice, but offers formal guarantees on the complexity of retained trajectories. We describe our approach in detail and provide empirical evidence of improved complexity over previous approaches.

2 Background and Preliminaries

2.1 Previous Pipeline Approaches

We describe prior synthetic data generation pipelines for tool-calling benchmarks from a modular perspective. At a high level, the pipeline comprises three components: a **Seeder**, a **Generator**, and a **Post-processor**. The **seeder** selects a tool-use scenario and the associated tool set; the **generator** then produces a query along with a candidate solution trajectory; and the **post-processor** validates the generated samples to ensure that only valid tool-execution trajectories are included in the final dataset.

Seeder. The seeder grounds generation in a specific tool set, domain, and/or use case — preventing model overload from excessive tools, avoiding incoherent queries from unrelated APIs, and enabling

systematic diversity control. While ToolQA (Zhuang et al., 2023) relies on curated query templates, most approaches such as ToolAlpaca (Tang et al., 2023), ToolEyes (Ye et al., 2024), ToolACE (Liu et al., 2024a), Open Functions (Patil et al., 2024; Charlie Cheng-Jie Ji, 2024), and APIBank (Li et al., 2023) maintain large API collections from which a domain-specific subset is sampled per datapoint.

Generator. The generator produces a query and its corresponding solution trajectory — a sequence of function calls that solve the query. In multi-turn settings, this is repeated across conversational turns, each building on prior context. ToolQA (Zhuang et al., 2023) uses manually crafted templates, while most approaches (Charlie Cheng-Jie Ji, 2024; Qin et al., 2023; Liu et al., 2024b) rely on LLMs, sometimes within multi-agent systems (Liu et al., 2024a; Tang et al., 2023; Prabhakar et al., 2025). Since LLMs are not error-free, this stage may yield erroneous trajectories.

Post-processor. This module retains only valid query-solution trajectories. Most approaches employ LLMs as validators (Li et al., 2023; Liu et al., 2024b;a; Chen et al., 2025), discarding invalid datapoints. For example, ToolACE (Liu et al., 2024a) uses a dual-layer system combining rule-based syntactic checks with model-based semantic verification, while APIGen (Liu et al., 2024b) assesses format consistency, executability, and semantic correctness. Other approaches use rule-based verification; Gorilla (Patil et al., 2024), for instance, applies AST-based matching to detect hallucinated function names.

In most prior approaches, synthetic data generation reduces to sampling a trajectory from the **Generator** distribution and retaining it only if it passes **Post-processor** verification — meaning the distribution of valid trajectories is proportional to the product of the generator distribution and the validity probability. This observation motivates the theoretical results in subsequent sections, for which we now introduce the necessary definitions and assumptions.

2.2 Formalization: Definition, Assumptions and Models

In this section, we formalize the concept of trajectory and define key elements to illustrate how filtering as a post-processing step can produce a light-tailed distribution. We also present and justify the key models and assumptions used in our analyses.

2.2.1 Trajectory Representation and Complexity

LLMs typically generate tool-calling trajectories as sequences of function calls. These sequences induce a directed acyclic graph (DAG) via data dependencies: a directed edge (v_i, v_j) indicates that the output of v_i is required to construct an input to v_j. Since any topological ordering of this graph yields an equivalent sequence, we represent trajectories as graphs.

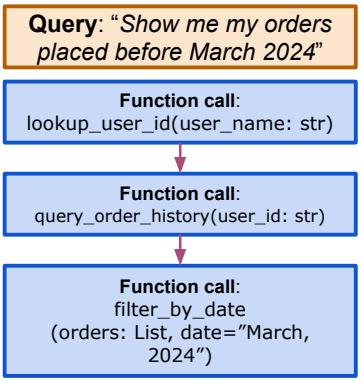

Figure 2: **Trajectory Representation and Complexity**: A trajectory $\mathcal{G} = (V, E)$ represented as a DAG, where nodes denote individual function calls and edges encode data dependencies. The three sequentially dependent calls yield a complexity of $c(G) = |V| = 3$ per definition 2.1.

Formally, we represent a trajectory τ as a DAG $G(\tau) = (V(\tau), E(\tau))$, where nodes $V(\tau) = \{v_1, v_2, \dots\}$ correspond to individual function calls and edges $E(\tau) \subseteq V(\tau) \times V(\tau)$ encode data dependencies. Each node v_i invokes a function with signature ϕ from the tool library $\mathcal{T}$, which specifies the tool's functionality, input arguments (including schema), and outputs. When τ is clear from context, we write $G = (V, E)$. See fig. 2 for an example. Refer to fig. 2 for a concrete example of trajectory representation.

Definition 2.1 (Trajectory Complexity)**.** In our analysis, we define the complexity of a trajectory $G = (V, E)$ as $c(G) = |V|$, the number of function calls.

This is a natural measure: it counts the total number of function invocations required to fulfill a user request. For the example shown in fig. 2, which involves looking up a user ID, querying their order history, and filtering by date, results in $c = 3$. A trajectory with $c = 50$ might involve orchestrating dozens of dependent calls across multiple services. While we stick to this simplistic definition of trajectory complexity, the subsequent analysis holds true for any $c(G) \in \Theta(|V|)$.

Definition 2.2 (Initial Trajectory Distribution)**.** Let $\mathbb{P}_g(c)$ represent probability mass function of the complexity of the generated trajectory. We define the CCDF (Complementary Cumulative Distribution Function) as $\bar{F}_g(c) = \sum_{k=c+1}^{\infty} \mathbb{P}_g(k)$.

Referring back to the section 2.1, the initial trajectory distribution is determined by the **seeder** and the **generator** module of the pipeline.

2.2.2 Validity of Trajectories and Modeling Errors

A trajectory is **valid** if every node is correctly specified—arguments are accurate, the appropriate function is called with correct syntax, and execution succeeds. This validity criterion is adapted from APIGen (Liu et al., 2024b).

Dziri et al. hypothesize that LLMs solve compositional tasks by reducing computation graphs to linearized subgraph matching against training patterns, rather than systematic reasoning. That is, each node in the computation graph is resolved by matching it against similar subgraphs seen during training. This gives rise to two node-level quantities: a *correctness rate*—the probability that the matched pattern yields the correct computation at that node—and a *collision rate*—the probability that an incorrectly matched pattern nonetheless produces the correct output by coincidence. Together, these determine the likelihood of a node yielding the correct answer. Empirically, they find that overall accuracy decays rapidly with graph size, consistent with error accumulation across nodes.

We model tool-call trajectory generation analogously, as a computation DAG where each node must be correctly generated with respect to syntax, arguments, and execution. Unlike the answer-level evaluation in Dziri et al., we require every node to be *computationally* correct—a wrong function call cannot be redeemed by a fortuitous collision. Collision rate is therefore irrelevant in our setting, and correctness reduces to whether each node is accurately specified. This motivates Assumption 2.1, which models each node as independently correct with fixed probability r, yielding a validity probability that decays exponentially with trajectory length.

Assumption 2.1 (Independent Node Correctness)**.** Every node v_i in the trajectory of complexity c is generated correctly with probability $r \in (0, 1)$. The trajectory is valid only when all the nodes are valid. The probability that a trajectory of complexity c is valid is given by:

$$\mathbb{P}_v(c) = r^c \tag{1}$$

Consequently, the probability that at least one node is incorrect is given by $1 - \mathbb{P}_v(c) = 1 - r^c$.

Given the initial distribution of trajectory complexity $\mathbb{P}_g(c)$ (refer definition 2.2) and the probability that a trajectory of given complexity is valid $\mathbb{P}_v(c)$ (refer assumption 2.1), we can compute the complexity distribution over valid trajectories. For synthetic generation pipelines that filter invalid trajectories, the output distribution over valid trajectories is:

$$\mathbb{P}_{g|v}(c) = \frac{\mathbb{P}_g(c) \cdot \mathbb{P}_v(c)}{\sum_{c=1}^{\infty} \mathbb{P}_g(c) \cdot \mathbb{P}_v(c)} \tag{2}$$

2.2.3 Iterative Refinement Model

Since BRIT applies iterative refinement to correct errors, we introduce a probabilistic model for this process. We assume that refinement does not alter the complexity of trajectory G, but instead attempts to fix erroneous nodes in place.

Assumption 2.2 (Error correction model)**.** Each iteration of the refinement process independently corrects an incorrect node with the probability $\rho \in (0, 1)$, labelled as the per-error fix rate. Errors at different nodes are corrected independently across both nodes and iterations.

Definition 2.3 (Refinement Budget)**.** The refinement budget $n : \mathbb{N} \to \mathbb{N}$ specifies the maximum number of iterations applied to each node in a trajectory with complexity c. We consider three regimes: $n(c) = 0$ (no refinement, pure filtering), $n(c) = n_0$ for a fixed constant n_0 (fixed budget) and $n(c)$ varying with c (adaptive budget).

Lemma 2.1 (Acceptance Probability under refinement)**.** *Under assumptions 2.1 and 2.2, the probability that a trajectory of complexity c would be valid after $n(c)$ refinement iterations is:*

$$Q_{n(c)}(c) = \Big(1 - (1 - r)(1 - \rho)^{n(c)}\Big)^c \tag{3}$$

The proof for this lemma is provided in Appendix A.1.

2.2.4 TEST FOR HEAVY TAILED DISTRIBUTION

We enumerate the different (equivalent) tests provided by Nair et al. (2022) to test whether a distribution is heavy tailed.

Definition 2.4. A distribution with CCDF $\bar{F}$ is said to be heavy tailed if and only if for all $\mu > 0$,

$$\limsup_{x \to \infty} \frac{\bar{F}(x)}{e^{-\mu x}} = \infty$$

Otherwise, the distribution is called light-tailed. Equivalently, the following test is also a necessary and a sufficient criteria for a heavy tailed distribution: The moment generating function $M(s) = \mathbb{E}[e^{sx}] = \infty$ for all $s > 0$.

3 THEORETICAL RESULTS

3.1 NO REFINEMENT RESULTS IN LIGHT-TAILED COMPLEXITY DISTRIBUTION

Theorem 3.1. *Under assumption 2.1, the output complexity distribution over valid trajectories is light-tailed when iterative refinement is absent ($n(c) = 0$).*

Proof. Refer Appendix A.2. □

Notably, the above result holds even if $\mathbb{P}_g(c)$ is heavy-tailed. This implies that improving the trajectory generation procedure without incorporating appropriate refinement would still result in a light-tailed distribution. Importantly, if validity probability decays exponentially with complexity, the resulting distribution is light-tailed regardless of the initial complexity distribution.

3.2 FIXED-BUDGET REFINEMENT YIELDS A LIGHT-TAILED COMPLEXITY DISTRIBUTION

Theorem 3.2. *Under assumptions 2.1 and 2.2, the output complexity distribution over valid trajectories is light-tailed when iterative refinement uses a fixed budget $n(c) = n_0$.*

Proof. Refer Appendix A.3. □

The theorem above establishes that under certain assumptions, applying iterative refinement for a fixed number of iterations is insufficient to make the output distribution heavy-tailed. Following the refinement model in section 2.2.3, our analysis considers only errors arising from incorrect syntax, input arguments, or execution — not missing function calls. In practice, trajectories may also omit required calls, which would further reduce acceptance probability. Notably, this would only strengthen the conclusion of the theorem, as the filtering distribution would decay even faster, making the light-tailed result more pronounced. This raises the question: Can adaptive refinement budget produce a heavy-tailed distribution under the stated assumptions? We address this question in section 3.3.

3.3 ADAPTIVE REFINEMENT PRESERVES HEAVY TAILS

Theorem 3.3. *Let the initial trajectory complexity distribution $\mathbb{P}_g$ be heavy-tailed. Under assumptions 2.1 and 2.2, if the refinement follows an adaptive budget according to $n(c) = \lceil \theta \log c \rceil$ where $\theta > 0$ is a constant, then the output distribution over valid trajectories is heavy-tailed.*

Proof. Refer Appendix A.4 □

This indicates that by designing a refinement budget that scales logarithmically with complexity, one can achieve a heavy-tailed complexity distribution over valid trajectories. However, the initial distribution must also exhibit this heavy tail for this approach to be effective.

4 EMPIRICAL EVALUATION OF BRIT

While the theoretical results in section 3 demonstrate that iterative refinement can produce heavy-tailed complexity distributions under specific assumptions, these results hold only when the initial distribution is already heavy-tailed. However, existing data generation approaches do not naturally produce such distributions—they tend to bias toward low-complexity examples. Therefore, to realize the benefits of iterative refinement predicted by our theory, we must first address this fundamental limitation by improving the initial distribution itself.

In this section, we empirically validate that the design choices in BRIT—particularly back-translation—shift the initial distribution toward higher complexity across various metrics, thereby creating the conditions under which iterative refinement can be effective. We then show empirically that (i) iterative refinement further increases complexity beyond back-translation alone (appendix B.1), and an adaptive budget retains most of this gain at lower compute (appendix B.2); (ii) the resulting data exhibits a heavier upper tail than the baselines (appendix B.3) at a moderate generation cost (appendix B.4); (iii) the added complexity reflects genuine dependencies and faithful queries rather than padding (appendix B.5); and (iv) the gains persist under a fixed backbone without refinement and grow with a stronger generator (appendices B.6 and B.7). This section is organized as follows: we describe our implementation approach in section 4.1, detail the trajectory validity evaluation process in section 4.2, discuss the baselines in section 4.3 and present the main empirical results in section 4.4 and the summary of additional analyses in section 4.5.

4.1 IMPLEMENTATION DETAILS OF BRIT

BRIT builds on the standard pipeline described in section 2.1, incorporating two key modifications: a **back-translation strategy** for trajectory generation and **LLM-based iterative refinement** to improve retention of complex samples. The **seeder** samples function signatures from a coherent theme; the **generator** employs back-translation to produce complex trajectories; and the **post-processor** evaluates samples for semantic validity, data-flow consistency, syntax correctness, and execution soundness, providing feedback for iterative refinement. After a fixed number of iterations, only valid trajectories are retained.

4.1.1 SEEDER

We use APIs collected by ToolLLM (Qin et al., 2023) from RapidAPI, hierarchically organized into **categories** (e.g., finance, education, legal, cryptocurrency) and **collections** of functionally similar APIs within each category. API execution is simulated using an LLM-based simulator from Guo et al., which takes a function definition, description, and input/output schema as input, returning a valid JSON response for well-formed requests or a structured error otherwise. We use `gpt-4o` as the backbone. During generation, the seeder samples 2 collections from a randomly selected category and passes all constituent function signatures to the generator.

4.1.2 GENERATOR

BRIT implements the Generator as a multi-agent system comprising a **Planner** and an **Executor** using `gpt-4o` powered agents. Given the tools provided by the seeder, the Planner generates a realistic and practical execution flow in natural language, consistent with the seeded context. The Executor then calls the necessary functions to verify that the generated plan is executionally feasible. Notably, the Executor produces the trajectory as a concrete sequence of function calls, while the Planner generates a human-readable intermediate plan that bridges the gap between high-level intent and executable calls. Structurally, the plan looks like a sequence of instructions, where each instruction names the target API, describes how to call it (using static arguments or results from prior function calls), and explains the underlying intent behind these instructions.

4.1.3 POST-PROCESSOR

The post-processor, powered by `gpt-4o`, evaluates trajectories along two dimensions: **semantic validity** (whether the trajectory faithfully executes each task in the Planner's output) and **data-flow consistency** (whether each function call uses the correct arguments from prior results). Errors across

both dimensions are consolidated into a single feedback message sent to the Planner, which revises its plan accordingly. After a fixed number of iterations, the final trajectory and plan are passed to an LLM to generate the associated query, forming the complete datapoint.

4.2 Validity Assessment

To validate generated datapoints, we adopt the three-stage verification approach from APIGen (Liu et al., 2024b). This process ensures that each query-trajectory pair meets high-quality standards. **(a) Format Checker**: Validates that generated data conforms to strict JSON structure with required fields ("query," "answer," and optionally "thought"). Datapoints with malformed entries or syntax errors are flagged and discarded. **(b) Execution Checker**: Executes well-formatted function calls using the simulator described in 4.1.1. Failed executions are filtered out. **(c) Semantic Checker**: Uses an LLM to verify that execution results semantically align with the original query intent. This stage identifies and filters cases where execution succeeds technically but produces meaningless results due to infeasible queries or incorrect argument choices. It evaluates: (1) alignment between function calls and the stated intent, (2) appropriateness of chosen functions, (3) correctness of the function call count, and (4) relevance of execution results to the query objective. For all data generation approaches, including ours, we apply this validation step to retain only valid datapoints for complexity comparison.

4.3 Baselines and Complexity Metrics

We compare against the following approaches for synthetic data generation: **(a) APIBank** (Li et al., 2023): Constructs tool-use dialogues by prompting LLMs to generate diverse API-related instructions and manually annotating their solution trajectories. We evaluate on the level-3 split, which contains the most complex manually curated query-trajectory pairs requiring multi-step planning and execution with interdependent function calls. **(b) APIGen** (Liu et al., 2024b): Generates query-answer pairs by sampling APIs and prompting LLMs to synthesize function calls. Verification occurs through three hierarchical stages: format checking, actual function executions, and semantic verification, with invalid instances filtered out. **(c) OpenFunctions-v2** (Charlie Cheng-Jie Ji, 2024): Fine-tunes LLMs on function-calling data supporting multiple languages and API types. Data generation involves prompting models with function definitions and natural language instructions to generate properly formatted function calls. **(d) ToolBench** (Qin et al., 2023): Collects 16,464 real-world REST APIs and uses ChatGPT to automatically generate diverse instructions covering both single-tool and multi-tool scenarios, with a depth-first search-based algorithm to annotate valid solution paths. For each baseline, we observe that they directly generate the query and do not involve iterative refinement.

Metric	Approach	Mean	Std	Median	IQR
# Unique Functions	APIBank	3.28	0.45	3	1
	APIGen	1.39	0.59	1	1
	OpenFunctions-v2	1.00	0.00	1	0
	ToolBench	3.54	0.96	3	1
	BRIT	**5.80**	**1.60**	**6**	**2**
	BRIT (SF)	5.61	1.47	6	1
	BRIT (Direct)	5.10	1.32	5	1
# Function Calls	APIBank	4.90	1.42	4	2
	APIGen	1.69	0.87	2	1
	OpenFunctions-v2	1.02	0.17	1	0
	ToolBench	4.13	1.08	4	2
	BRIT	**7.43**	**5.61**	**6**	**3**
	BRIT (SF)	7.16	6.15	6	2
	BRIT (Direct)	6.65	6.09	6	2
Longest Path Length	APIBank	2.02	0.97	2	2
	APIGen	0.52	0.50	1	1
	OpenFunctions-v2	0.01	0.10	0	0
	ToolBench	1.39	0.81	1	1
	BRIT	**2.62**	**1.63**	**2**	**1**
	BRIT (SF)	2.49	1.63	2	2
	BRIT (Direct)	2.61	2.18	2	1
# Edges	APIBank	5.16	3.37	4	5.75
	APIGen	0.67	0.86	1	1
	OpenFunctions-v2	0.01	0.17	0	0
	ToolBench	3.12	2.21	3	2
	BRIT	**11.60**	**22.80**	**7**	**5.25**
	BRIT (SF)	11.17	25.82	6	5.00
	BRIT (Direct)	10.31	16.40	6	5
Cyclomatic Complexity	APIBank	2.72	1.88	2	4
	APIGen	0.52	0.50	2	4
	OpenFunctions-v2	0.01	0.10	0	0
	ToolBench	1.58	1.36	1	1
	BRIT	**6.70**	**17.95**	**3**	**4**
	BRIT (SF)	6.54	20.43	3	4
	BRIT (Direct)	6.07	21.17	3	4

Table 1: Complexity metrics across different data generation approaches (SF denotes Simple Feedback).

To assess the complexity of generated trajectories, we use the following metrics: **(a)** Number of unique functions, **(b)** Number of function calls, **(c)** Number of edges in longest dependency path, **(d)** Number of dependency edges and **(e)** Cyclomatic complexity (measuring the number of linearly independent paths from the first function call). For each statistic, we report the mean, standard

deviation, median and Inter-quartile Range (IQR). Complexity metrics are evaluated on a random sample of 1000 datapoints per dataset.

4.4 Complexity of BRIT and Baselines

For our approach, we apply 5 iterations of refinement and evaluate two ablations: **(a) Direct**: Generates the query first, then the trajectory, without back-translation. **(b) Simple Feedback**: Applies refinement using only semantic validity checks, omitting data flow consistency checks. While theory predicts that logarithmic refinement budgets preserve heavy-tailed distributions, our empirical focus is on demonstrating that BRIT achieves improved complexity. Therefore, we use a fixed refinement budget (refer section 4.5 for adaptive budget results). Table 1 shows that BRIT achieves the highest complexity across most metrics. The DIRECT ablation, which generates queries before trajectories, yields significantly lower complexity values, validating the importance of back-translation. Omitting data flow consistency checks further reduces refinement effectiveness, resulting in lower final complexity values.

4.5 Summary of Additional Empirical Analyses

Appendix B reports supplementary analyses of BRIT, summarized below. **Refinement iterations** (Appendix B.1): Increasing the number of refinement iterations broadly raises mean complexity across metrics, consistent with the higher acceptance probability predicted by Lemma 2.1. **Adaptive budget** (Appendix B.2): An adaptive budget $n(c) = \lceil \theta \ln c \rceil$ retains 85% of the fixed budget's high-complexity yield at 63% of its refinement compute. **Tail diagnostics** (Appendix B.3): BRIT attains substantially lower Hill tail indices and maxima up to two orders of magnitude larger than the baselines, suggesting a heavier upper tail. **Sample efficiency and cost** (Appendix B.4): Refinement more than doubles acceptance at roughly twice the per-accepted-sample cost of filter-and-resample, and approximately 60% of accepted trajectories with $|V| \geq 8$ are valid only after refinement. **Redundancy and alignment** (Appendix B.5): BRIT trajectories contain few duplicate or unconsumed calls, and their back-translated queries achieve higher alignment scores than the ablations and comparably complex baselines. **No refinement** (Appendix B.6): Under a fixed `gpt-4o` backbone and without refinement, our generation procedures produces trajectories roughly 3 to 8 times more complex than re-implemented baselines. **Stronger backbone** (Appendix B.7): Upgrading the generator to `gpt-5.5` increases the complexity of BRIT on every metric, while query-first single-shot baselines with a better generation backbone remain limited to one or two function calls.

5 Additional Discussions

Can BRIT generate datapoints for training? To assess whether synthetic datapoints from BRIT can enhance downstream tool-calling performance, we evaluate on two comprehensive benchmarks: **APIBank**[1] (Li et al., 2023) and **ACEBench**[2] (Chen et al., 2025). APIBank's manually annotated evaluation set spans three complexity levels reflecting distinct tool usage abilities: **level-1**: basic API calling with known APIs, **level-2**: retrieval and calling with unknown APIs, and **level-3** multi-step planning with unknown APIs. ACEBench provides three evaluation categories — **Normal** (basic scenarios), **Special** (ambiguous or incomplete instructions), and **Agent** (multi-agent multi-turn interactions) — enabling assessment across varying interaction patterns. This combination allows us to test whether higher-complexity synthetic data translates to improved performance across varying task difficulty levels and interaction patterns. As the test performance on a given set may depend on the alignment between training and test complexity distributions, we construct three synthetic training splits generated from BRIT of 500 datapoints each with progressively increasing trajectory complexity. The **easy-split** contains trajectories with 1–3 function calls, the **medium-split** contains 4–6 function calls, and the **difficult-split** contains more than 6 function calls.

The results in table 2 show that finetuning on simpler splits yields superior performance on level-2 and level-2, while finetuning on harder splits substantially improves level-3 performance, particularly for `Llama3.1-8b-instruct` (Grattafiori et al., 2024). This pattern reflects the distributional alignment between training and evaluation: since level-1 involves only single API calls and level-2

[1]Licensed under Apache License

[2]Licensed under MIT License

Training Data	Level 1	Level 2	Level 3
`Llama3.1-8b-instruct`			
N/A	71.7	38.5	33.9
Easy split (1-3)	**76.2**	**44.4**	32.6
Medium split (4-6)	73.4	41.5	34.3
Difficult split (> 6)	74.7	42.2	**38.4**
`Qwen3-8b`			
N/A	73.7	44.4	34.7
Easy split (1-3)	**75.7**	**43.0**	46.5
Medium split (4-6)	75.2	42.2	46.1
Difficult split (> 6)	74.7	41.5	**47.3**

Table 2: **APIBank Results**: Performance comparison between no additional training (N/A) and finetuning on easy, medium, and difficult splits.

Training Data	Normal	Special	Agent	Overall
`llama3.1-8b-instruct`				
N/A	47.2	5.3	4.2	29.3
Easy split (1-3)	47.2	7.3	6.7	**30.3**
Medium split (4-6)	44.2	7.3	6.7	28.5
Difficult split (> 6)	43.8	4.0	6.7	27.4
`Qwen3-8b`				
N/A	74.7	79.3	0.0	64.4
Easy split (1-3)	75.2	80.7	0.0	**65.1**
Medium split (4-6)	73.4	79.3	0.0	63.6
Difficult split (> 6)	73.8	78.0	0.0	63.5

Table 3: **ACEBench Results**: Performance comparison between no additional training (N/A) and finetuning on easy, medium, and difficult splits.

corresponds to retrieval followed by a single call—both requiring fewer than two effective function calls—models trained on easier splits better capture these patterns. Conversely, level-3 performance benefits from exposure to complex multi-step trajectories, indicating that harder splits provide the necessary complexity for this setting. These results underscore a practical trade-off: while BRIT can generate much more complex trajectories, selecting training data that aligns with the target task's complexity is essential for optimal downstream performance.

For ACEBench, as shown in table 3, finetuning on the easy split yields the most consistent performance gains across models. This pattern reflects the nature of ACEBench's evaluation categories: the majority of datapoints—whether in Normal, Special, or Agent splits—involve relatively simple scenarios with single function calls or limited multi-turn complexity. Even in multi-turn conversations, individual user queries typically map to a single function call. Consequently, training on simpler trajectories from our approach aligns better with the test distribution, as evidenced by the improvements in Qwen3-8b (Yang et al., 2025a) (64.4 $\rightarrow$ 65.1) and Llama-3.1-8b-instruct (29.3 $\rightarrow$ 30.3). Conversely, finetuning on more complex splits introduces distributional mismatch: models trained on trajectories with substantially more function calls than appear in the test set exhibit degraded performance, suggesting that exposure to out-of-distribution complexity harms generalization to the actual target tasks. Additionally, the synthetic training data involves functions and APIs that are distributionally different from those in the test set, which further influences downstream performance and may explain the modest overall improvements even when complexity is well-aligned.

We provide additional discussion on *related advances in synthetic tool-calling data generation*, *factors beyond trajectory complexity that influence task difficulty*, *the practical validity of our theoretical assumptions*, and *the applicability of our findings to compositional problem solving* more broadly. We refer the reader to Appendix C for details.

6 Conclusion

This paper addresses a fundamental bias in synthetic data generation for tool-calling benchmarks: existing approaches systematically produce low-complexity examples. We identify two key culprits—direct query synthesis and filtering as post-processing—and provide theoretical analysis showing that filtering necessarily yields light-tailed complexity distributions. We propose BRIT, which modifies both stages: back-translation in the generator explicitly optimizes complexity, while iterative refinement in the post-processor preserves complex valid examples. Theoretically, we show that logarithmic refinement budgets can recover heavy-tailed distributions. Empirically, BRIT achieves substantially higher complexity across multiple metrics compared to baselines, and training on complexity-aligned data improves downstream performance. This work demonstrates the importance of systematically including high-complexity instances in tool-calling evaluation and provides a scalable method to achieve this goal.

AI USE STATEMENT

The core ideas, framework design, experimental design, and methodology implementation in this work were developed entirely by the authors without using generative AI. We used generative AI tools for paraphrasing and language polishing of the manuscript text, in implementing some analyses of our method and as a backbone in our generation methodology. All AI-assisted text edits and code-implementations were reviewed by the authors for correctness. We take full responsibility for the final content of this work, including all text, claims, and artifacts.

REFERENCES

Mohammad Abdollahi, Khandaker Rifah Tasnia, Soumit Kanti Saha, Jinqiu Yang, Song Wang, and Hadi Hemmati. Demystifying errors in llm reasoning traces: An empirical study of code execution simulation. *arXiv preprint arXiv:2512.00215*, 2025.

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, et al. Agentharm: A benchmark for measuring harmfulness of llm agents. *arXiv preprint arXiv:2410.09024*, 2024.

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

Fanjia Yan Shishir G. Patil Tianjun Zhang Ion Stoica Joseph E. Gonzalez Charlie Cheng-Jie Ji, Huanzhi Mao. Gorilla openfunctions v2. 2024.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Yuefeng Huang, et al. Acebench: Who wins the match point in tool usage? *arXiv preprint arXiv:2501.12851*, 2025.

Mingyang Chen, Haoze Sun, Tianpeng Li, Fan Yang, Hao Liang, Keer Lu, Bin Cui, Wentao Zhang, Zenan Zhou, and Weipeng Chen. Facilitating multi-turn function calling for llms via compositional instruction tuning. *arXiv preprint arXiv:2410.12952*, 2024.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, et al. Faith and fate: Limits of transformers on compositionality. *Advances in neural information processing systems*, 36:70293–70332, 2023.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1), 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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

Bruce M Hill. A simple general approach to inference about the tail of a distribution. *The annals of statistics*, pp. 1163–1174, 1975.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*, 2023.

Fabrizio Lillo and Salvatore Ruggieri. Estimating the total volume of queries to a search engine. *IEEE Transactions on Knowledge and Data Engineering*, 34(11):5351–5363, 2021.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, et al. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*, 2024a.

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

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, 2025.

Hanjun Luo, Shenyu Dai, Chiming Ni, Xinfeng Li, Guibin Zhang, Kun Wang, Tongliang Liu, and Hanan Salam. Agentauditor: Human-level safety and security evaluation for llm agents. *arXiv preprint arXiv:2506.00641*, 2025.

Seiji Maekawa, Jackson Hassell, Pouya Pezeshkpour, Tom Mitchell, and Estevam Hruschka. Towards reliable benchmarking: A contamination free, controllable evaluation framework for multi-step llm function calling. *arXiv preprint arXiv:2509.26553*, 2025.

Yao Mu, Qinglong Zhang, Mengkang Hu, Wenhai Wang, Mingyu Ding, Jun Jin, Bin Wang, Jifeng Dai, Yu Qiao, and Ping Luo. Embodiedgpt: Vision-language pre-training via embodied chain of thought. *Advances in Neural Information Processing Systems*, 36:25081–25094, 2023.

Jayakrishnan Nair, Adam Wierman, and Bert Zwart. *The fundamentals of heavy tails: Properties, emergence, and estimation*, volume 53. Cambridge University Press, 2022.

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

Steven T Piantadosi. Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic bulletin & review*, 21(5):1112–1130, 2014.

David MW Powers. Applications and explanations of zipf's law. In *New methods in language processing and computational natural language learning*, 1998.

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

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

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

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*, 2023.

Jun Wang, Jiamu Zhou, Muning Wen, Xiaoyun Mo, Haoyu Zhang, Qiqiang Lin, Cheng Jin, Xihuai Wang, Weinan Zhang, and Qiuying Peng. Hammerbench: Fine-grained function-calling evaluation in real mobile device scenarios. *arXiv preprint arXiv:2412.16516*, 2024.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Berkeley function calling leaderboard, 2024.

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

Chen Yang, Ran Le, Yun Xing, Zhenwei An, Zongchao Chen, Wayne Xin Zhao, Yang Song, and Tao Zhang. Toolmind technical report: A large-scale, reasoning-enhanced tool-use dataset. *arXiv preprint arXiv:2511.15718*, 2025b.

Seungbin Yang, ChaeHun Park, Taehee Kim, and Jaegul Choo. Can tool-augmented large language models be aware of incomplete conditions? *arXiv preprint arXiv:2406.12307*, 2024.

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

Junjie Ye, Guanyu Li, Songyang Gao, Caishuang Huang, Yilong Wu, Sixian Li, Xiaoran Fan, Shihan Dou, Qi Zhang, Tao Gui, et al. Tooleyes: Fine-grained evaluation for tool learning capabilities of large language models in real-world scenarios. *arXiv preprint arXiv:2401.00741*, 2024.

Fan Yin, Zifeng Wang, I-Hung Hsu, Jun Yan, Ke Jiang, Yanfei Chen, Jindong Gu, Long Le, Kai-Wei Chang, Chen-Yu Lee, et al. 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)*, pp. 32600–32616, 2025.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823*, 2023.

Yuxiang Zhang, Jing Chen, Junjie Wang, Yaxin Liu, Cheng Yang, Chufan Shi, Xinyu Zhu, Zihao Lin, Hanwen Wan, Yujiu Yang, et al. Toolbehonest: A multi-level hallucination diagnostic benchmark for tool-augmented large language models. *arXiv preprint arXiv:2406.20015*, 2024.

Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, Jie Jiang, and Bin Cui. Retrieval-augmented generation for ai-generated content: A survey. *arXiv preprint arXiv:2402.19473*, 2024.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems*, 36: 50117–50143, 2023.

A PROOFS

The proofs for Lemma 2.1, Theorem 3.1, Theorem 3.2, and Theorem 3.3 are provided below.

A.1 PROOF FOR LEMMA 2.1

Proof. A trajectory $G = (V, E)$ would be accepted when each of the nodes are valid. A node $v_i \in V$ would be ultimately correct if either (a) it was initially correct with the probability r or (b) it was initially incorrect with probability $(1 - r)$ but was rectified in one of the refinement iterations. With probability $(1 - \rho)^{n(c)}$, the corresponding node would not be rectified in any of the iterations and hence, with probability $1 - (1 - \rho)^{n(c)}$ it will be rectified in one of the iterations. Thus, the total probability that v_i would be correct is:

$$\mathbb{P}(v_i \text{ is correct}) = r + (1 - r) \times \left(1 - (1 - \rho)^{n(c)}\right)$$

$$\mathbb{P}(v_i \text{ is correct}) = \left(1 - (1 - r)(1 - \rho)^{n(c)}\right)$$

$Q_{n(c)}(c)$ is the probability when all the nodes are correct, which is equal to the above expression raised to c owing to the assumptions 2.1 and 2.2. □

A.2 Proof for Theorem 3.1

Proof. To show this, we use the criterion from section 2.2.4 that a distribution is light-tailed if its moment generating function is finite for some $s > 0$. Below, we establish finiteness for all $s \in (0, -\log r)$ which is non-empty as $r \in (0, 1)$ as described in assumption 2.1. Referring to eq. (2), the output distribution under the stated assumption is:

$$\mathbb{P}_{g|v}(c) = \frac{\mathbb{P}_g(c) \cdot \mathbb{P}_v(c)}{Z_0}$$

where $Z_0 = \sum_{c=1}^{\infty} \mathbb{P}_g(c) \cdot \mathbb{P}_v(c)$. First, note that Z_0 is finite because:

$$\begin{aligned} Z_0 &= \sum_{c=1}^{\infty} \mathbb{P}_g(c) \cdot \mathbb{P}_v(c) \leq \sum_{c=1}^{\infty} 1 \cdot \mathbb{P}_v(c) \\ &\leq \sum_{c=1}^{\infty} r^c = \frac{r}{1-r} < \infty \end{aligned} \tag{4}$$

To get the above inequality, we used $\mathbb{P}_g(c) \leq 1$ for all c and the assumption 2.1. The moment generating function $M_{g|v}(s)$ for the distribution in eq. (2) is given as follows:

$$\begin{aligned} M_{g|v}(s) = \mathbb{E}_{g|v}[e^{sc}] &= \frac{\sum_{c=1}^{\infty} \mathbb{P}_g(c) \cdot \mathbb{P}_v(c) \cdot e^{sc}}{Z_0} \\ &= \frac{\sum_{c=1}^{\infty} \mathbb{P}_g(c) \cdot (re^s)^c}{Z_0} \end{aligned} \tag{5}$$

For any $s < -\log r$, we have $re^s < 1$. This means:

$$\begin{aligned} M_{g|v}(s) &\leq \frac{\sum_{c=1}^{\infty} 1 \cdot (re^s)^c}{Z_0} \\ &= \frac{re^s}{Z_0\,(1 - re^s)} < \infty \end{aligned} \tag{6}$$

This indicates that the moment generating function is finite for all $s \in (0, -\log r)$ and therefore, the distribution over valid trajectories without refinement is light-tailed. □

A.3 Proof for Theorem 3.2

Proof. The proof followed the same strategy as theorem 3.1. Using the notation from lemma 2.1, the output complexity distribution over valid trajectories is given by:

$$\mathbb{P}_{g|v,n(c)}(c) = \frac{\mathbb{P}_g(c) \cdot Q_{n(c)}(c)}{\sum_{k=1}^{\infty} \mathbb{P}_g(k) \cdot Q_{n(c)}(k)}$$

Similar to the computations in eq. (4), we can show that the denominator $Z_{n(c)} = \sum_{c=1}^{\infty} \mathbb{P}_g(c) \cdot Q_{n(c)}(c)$ is bounded when $n(c) = n_0$. We define $r_{n(c)}$ as follows:

$$r_{n(c)} = 1 - (1 - r)(1 - \rho)^{n_0} \tag{7}$$

Using eq. (3) and eq. (7), we can express the moment generating function for the distribution $\mathbb{P}_{g|v,n_0}$ where $n(c) = n_0$ for all c as follows:

$$M_{g|v,n_0}(s) = \frac{\sum_{c=1}^{\infty} \mathbb{P}_g(c) \cdot (r_{n_0} e^s)^c}{Z_{n_0}} \tag{8}$$

This form is very similar to the computations in eq. (5) from the theorem 3.1. Thus, it follows that the moment would be finite for all $s \in (0, -\log r_{n_0})$ and therefore, the distribution over valid trajectories with fixed-budget refinement is still light-tailed.

□

A.4 Proof for Theorem 3.3

Proof. Similar to the strategy employed for showing theorem 3.2, we evaluate whether the moment generating function for $\mathbb{P}_{g|v,n(c)}$ is unbounded for all $s > 0$. Recall that,

$$\mathbb{P}_{g|v,n(c)}(c) = \frac{\mathbb{P}_g(c) \cdot Q_{n(c)}(c)}{\sum_{k=1}^{\infty} \mathbb{P}_g(k) \cdot Q_{n(k)}(k)}$$

To show that the above distribution is heavy-tailed, we need to show that:

$$M_{g|v,n(c)}(s) = \frac{1}{Z_{n(c)}} \cdot \sum_{c=1}^{\infty} e^{sc} \cdot \mathbb{P}_g(c) \cdot Q_{n(c)}(c)$$

is unbounded for all $s > 0$ where $Z_{n(c)} = \sum_{c=1}^{\infty} \mathbb{P}_g(c) \cdot Q_{n(c)}(c) \leq \sum_{c=1}^{\infty} \mathbb{P}_g(c) \cdot 1 = 1 < \infty$. To approach this, we first attempt to lower bound $Q_{n(c)}(c)$. Define $e_{n(c)} = (1-r)(1-\rho)^{n(c)}$, then:

$$Q_{n(c)}(c) = \left(1 - e_{n(c)}\right)^c \tag{9}$$

Taking logs and using $\log(1-x) \geq -x/(1-x)$, we have

$$\log\left(Q_{n(c)}(c)\right) = c \log\left(1 - e_{n(c)}\right) \geq \frac{-ce_{n(c)}}{1 - e_{n(c)}}, \tag{10}$$

where the inequality holds for all $e_{n(c)} \in [0, 1)$. Given that $\theta \log c \leq \lceil \theta \log c \rceil \leq \theta \log c + 1$ and $(1-\rho) \in (0, 1)$, we have the following:

$$(1-\rho)^{\theta \log c + 1} \leq (1-\rho)^{\lceil \theta \log c \rceil} \leq (1-\rho)^{\theta \log c}$$

Substituting $\mu = -\log(1-\rho)$ and multiplying the above inequality with $(1-r)$, we have:

$$\begin{aligned} (1-r)(1-\rho)c^{-\mu\theta} \leq e_{n(c)} \leq (1-r)c^{-\mu\theta} \\ ac^{-\mu\theta} \leq e_{n(c)} \leq bc^{-\mu\theta} \end{aligned} \tag{11}$$

where $a = (1-r)(1-\rho)$ and $b = (1-r)$ are constants independent of c. Based on the bounds obtained on $e_{n(c)}$, we can lower bound the eq. (10) as follows:

$$\log\left(Q_{n(c)}\right) \geq \frac{-bc^{1-\mu\theta}}{1 - bc^{-\mu\theta}} \tag{12}$$

Now choose $c_0 = \lceil (2b)^{\frac{1}{\mu\theta}} \rceil$, so that, we have $1 - bc^{-\mu\theta} \geq \frac{1}{2}$. Then, for all $c \geq c_0$, we have the following:

$$\begin{aligned} \log\left(Q_{n(c)}\right) \geq -2bc^{1-\mu\theta} \\ Q_{n(c)} \geq e^{-2bc^{1-\mu\theta}} \end{aligned} \tag{13}$$

We consider two cases based on the value of $\mu\theta$: **(a)** $\mu\theta \geq 1$ and **(b)** $0 < \mu\theta < 1$. For the case (a), we have $c^{1-\mu\theta} \leq c^0 = 1$. Thus, $Q_{n(c)} \geq e^{-2bc^{1-\mu\theta}} \geq e^{-2b}$. Clearly, the lower bound is constant for all $c > c_0$. This means, for any $s > 0$:

$$M_{g|v,n(c)}(s) \geq \frac{1}{Z_{n(c)}} \cdot \left(\sum_{c=c_0}^{\infty} e^{sc} \cdot \mathbb{P}_g(c) \right) \times e^{-2b}$$

The term within the parenthesis is unbounded for any $s > 0$ as $\mathbb{P}_g$ is heavy-tailed. This implies that $M_{g|v,n(c)}(s)$ is unbounded for any $s > 0$ proving the claim for case (a).

For case (b), we have the following expression:

$$M_{g|v,n(c)}(s) \geq \frac{1}{Z_{n(c)}} \cdot \sum_{c=c_0}^{\infty} \mathbb{P}_g(c) \cdot e^{sc - 2bc^{1-\mu\theta}} \tag{14}$$

For the exponent of e: $sc - 2bc^{1-\mu\theta}$, we note that $2bc^{1-\mu\theta} = o(c)$. We can choose c_1 and $0 < \epsilon < s$ such that for all $c > c_1$, we have $2bc^{1-\mu\theta} < \epsilon c$. Thus, we can get the following lower bound for $M_{g|v,n(c)}(s)$:

$$M_{g|v,n(c)}(s) \geq \frac{1}{Z_{n(c)}} \cdot \sum_{c=\max(c_0,c_1)}^{\infty} \mathbb{P}_g(c) \cdot e^{(s-\epsilon)c} \tag{15}$$

As $(s - \epsilon) > 0$, we again have the sum unbounded by evaluating the moment generating function of $\mathbb{P}_g$ at $s - \epsilon$. Thus, for both the cases ((a) and (b)), the sum would be unbounded showing that the resultant distribution can preserve its heavy-tail. □

B Additional Analyses

Owing to space constraints in the main text, this section presents supplementary analyses of BRIT, including tail diagnostics, controlled comparisons, and cost accounting.

B.1 Influence of Refinement Iterations

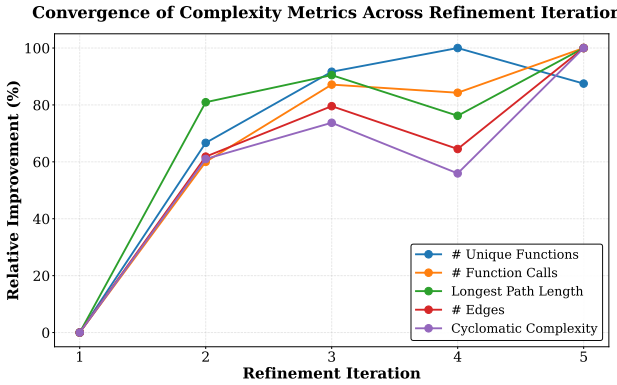

Figure 3: Influence of refinement iterations on different complexity metrics.

While section 4.4 justifies back-translation and sophisticated feedback as necessary for improved performance, this section empirically demonstrates how increasing refinement iterations enhances the mean values of our complexity metrics. For each metric, if the mean statistic at a given iteration is μ, we compute the relative improvement $\frac{\mu-\mu_{\min}}{\mu_{\max}-\mu_{\min}}$. fig. 3 showcases that broadly several complexity metrics are improved with the number of refinement iterations. Referring to lemma 2.1, we note that as $n(c)$, the refinement budget at a given complexity c, increases, the value of acceptance probability increases. Thus, more complex trajectories are accepted with a higher probability resulting in a broad increase across several complexity metrics.

B.2 Testing Adaptive Budget

While the experiments in section 4 use a fixed number of refinement iterations, here we evaluate an adaptive budget. For an output of complexity c, we set the maximum number of refinement iterations to $n(c) = \lceil \theta \ln c \rceil$. We consider a total of 3,650 instances generated by BRIT and use $n(c)$ as the maximum refinement budget (table 4). If an instance has already been rectified before $n(c)$, then we stop the refinement loop even if the sample-specific refinement budget is not completed.

Condition	% Valid	Avg. refinement rounds / datapoint	# Valid trajectories ($\lvert V \rvert \geq 8$)
Filtering ($n = 0$)	31.8	0.00	225
Adaptive ($\theta = 1$)	59.0	1.15	480
Fixed ($n = 5$)	72.4	1.84	565

Table 4: Acceptance, refinement cost, and high-complexity yield under different refinement budgets, obtained by replaying stored refinement iterations over 3,650 generated instances. The adaptive budget follows $n(c) = \lceil \theta \ln c \rceil$.

Although overall acceptance is highest under the fixed budget, acceptance of high-complexity trajectories (initial $c \geq 8$) rises from 17% under filtering to 58% under the adaptive budget ($\theta = 1$) and 62% under the fixed budget ($n = 5$). The adaptive budget thus retains 85% of the fixed budget's high-complexity yield while using 63% of its refinement compute, corresponding to roughly $1.36\times$ more high-complexity valid trajectories per refinement round.

B.3 TAIL DIAGNOSTICS

As means and standard deviations do not fully characterize tail behavior, we complement them with three distributional statistics: the 99th percentile $Q_{0.99}$, the maximum observed value, and a fitted tail index. We estimate the tail index with the Hill estimator (Hill, 1975). Suppose the complementary CDF of a complexity metric X decays as a power law, $\bar{F}(x) = \Pr(X > x) \sim L(x)\, x^{-\alpha}$, where L is slowly varying and $\alpha > 0$ is the tail index. Given N positive samples with order statistics $X_{(1)} \geq X_{(2)} \geq \cdots \geq X_{(N)}$, the Hill estimator computed from the k largest observations is

$$\hat{\alpha}_k = \left(\frac{1}{k} \sum_{i=1}^{k} \ln \frac{X_{(i)}}{X_{(k+1)}} \right)^{-1}. \tag{16}$$

A smaller $\hat{\alpha}$ indicates slower tail decay and hence a heavier tail. For reference, a power-law tail with $\alpha \leq 2$ has infinite variance, and one with $\alpha \leq 1$ has infinite mean. We exclude zero-valued samples, since the estimator requires positive values, and select k as [the top X% of samples / the stable region of the Hill plot].

Metric	BRIT			Baselines	
	$Q_{0.99}$	max	$\hat{\alpha}$	max	$\hat{\alpha}$
# Dependency edges ($\lvert E \rvert$)	84	463	1.54	≤ 15	3.0–5.4
# Function calls ($\lvert V \rvert$)	32	78	2.46	≤ 12	8.8–16.9
Cyclomatic complexity	38	353	1.22	≤ 6	6.2

Table 5: Tail diagnostics for BRIT and baselines. $Q_{0.99}$ denotes the 99th percentile and $\hat{\alpha}$ the Hill estimate of the tail index (lower indicates a heavier tail). Baseline columns report the largest maximum and the range of $\hat{\alpha}$ across baselines.

As shown in table 5, BRIT attains a substantially lower tail index than every baseline across all three metrics, indicating a heavier upper tail. Its maximum observed values exceed those of the baselines by one to two orders of magnitude for dependency edges and cyclomatic complexity, and by roughly $6\times$ for the number of function calls.

B.4 SAMPLE EFFICIENCY AND GENERATION COST

Approach	# Attempts	% Valid		Avg. refinement
		Initial	Post-refinement	rounds / datapoint
BRIT	3,650	31.8	72.4	1.01
BRIT (SF)	3,608	31.7	74.9	1.02
BRIT (Direct)	2,714	39.0	76.2	0.88

Table 6: Generation yield for BRIT and its ablations, computed over all generation attempts rather than only retained datapoints. Initial acceptance corresponds to pure filtering ($n = 0$); post-refinement acceptance is measured after at most five refinement rounds.

To assess sample efficiency, we rerun generation for BRIT and its two ablations while logging every attempt, rather than only the retained datapoints (table 6). Refinement roughly doubles acceptance for all three variants, raising it from 31.8% to 72.4% for BRIT.

In terms of cost, an instrumented run averages 66.5 LLM calls and 25.2 simulator calls (approximately 437K tokens) per attempted datapoint, or roughly 89 LLM calls per accepted datapoint. This is approximately twice the per-accepted-sample cost of naive filter-and-resample generation. In exchange, refinement yields $2.3\times$ more accepted datapoints and provides access to high-complexity trajectories that resampling rarely reaches.

The acceptance statistics suggest why. Without refinement, acceptance is biased toward simpler trajectories: raw attempts average $|V| \approx 7.7$, whereas those accepted by filtering alone average $|V| \approx 5.5$,

consistent with theorem 3.1 which shows filtering alone results in light-tailed complexity distribution. Refinement recovers many of the harder cases: approximately 60% of accepted trajectories with $|V| \geq 8$ are valid only after refinement. Matching this count through filter-and-resample would require roughly $2.5\times$ more raw generations, since resampling cannot repair a near-miss complex trajectory and must instead discard it. These results suggest that the gains stem from refinement itself rather than from additional compute alone.

B.5 Beyond Complexity: Redundancy and Alignment

Complexity metrics alone do not show whether the additional calls in BRIT trajectories reflect genuine dependencies or padding, nor whether the back-translated query faithfully describes its trajectory. We therefore evaluate redundancy and query–trajectory alignment.

Redundancy. We measure the **duplicate-call fraction**, the proportion of calls that repeat an earlier call with identical arguments, and the **unconsumed-output fraction**, the proportion of calls whose outputs are not used by any subsequent call. BRIT has a duplicate-call fraction of 0.5%, compared with 6.7% for ToolBench, and the lowest unconsumed-output fraction among all compared datasets (0.54, versus 0.66–1.00 for baselines). This suggests that the additional calls in BRIT trajectories predominantly reflect genuine data-flow dependencies rather than redundant steps.

Alignment. We score query–trajectory alignment with `gpt-5.5` on a 1–5 scale using 200 samples per approach. BRIT scores 3.71, above both of its ablations (BRIT (SF): 3.64; BRIT (Direct): 3.52) and the baselines of comparable complexity (ToolBench: 2.42; ToolACE: 3.15). On trajectory uniqueness, BRIT trails APIGen (4.72). We attribute this gap to APIGen's predominantly single-call trajectories, for which uniqueness is nearly guaranteed by construction, rather than to lower query fidelity in BRIT. Since these scores rely on an LLM judge, we interpret them as a relative comparison across approaches rather than an absolute measure of correctness.

B.6 Performance without Refinement

Approach	# Unique functions	# Calls ($\lvert V\rvert$)	Longest path	# Edges ($\lvert E\rvert$)
BRIT (TF)	**5.46**	**6.48**	**2.12**	7.46
Direct (QF)	4.66	6.34	2.05	**7.73**
APIGen	1.93	1.95	1.22	1.90
OpenFunctions-v2	1.00	1.00	0.98	0.98

Table 7: Complexity of valid trajectories from a single-agent pass without refinement. TF: trajectory-first (back-translation); QF: query-first. All approaches use `gpt-4o` as the generator with the same simulator and validity criteria; APIGen and OpenFunctions-v2 are re-implemented under this setup.

To attribute the complexity gains of BRIT to its generation strategy rather than to refinement or additional compute, we evaluate a single-agent pass without refinement. We compare it against APIGen and OpenFunctions-v2, re-implemented with the same `gpt-4o` generator, simulator, and validity criteria, so that the backbone is held fixed across all approaches (table 7).

Even without refinement, a single trajectory-first pass produces trajectories roughly 3 to 8 times more complex than the re-implemented baselines ($|V| = 6.48$ versus 1.95 and 1.00; $|E| = 7.46$ versus 1.90 and 0.98). The query-first single-agent variant attains comparable complexity ($|V| = 6.34$, $|E| = 7.73$), likely because its prompt also instructs the model to compose queries that are as complex as the available tools allow. This suggests that much of the gain over prior pipelines stems from explicitly targeting complexity at generation time, with back-translation contributing a modest further increase in unique functions and function calls.

Approach	gpt-4o				gpt-5.5			
	F	$\|V\|$	ℓ	$\|E\|$	F	$\|V\|$	ℓ	$\|E\|$
BRIT	**5.78**	**7.43**	**2.62**	**11.60**	**9.84**	**12.12**	**3.86**	23.21
BRIT (Direct)	5.10	6.65	2.61	10.31	6.04	9.88	3.78	**24.32**
APIGen	1.93	1.95	1.22	1.90	2.04	2.08	1.12	2.15
OpenFunctions-v2	1.00	1.00	0.98	0.98	1.00	1.00	0.97	0.97

Table 8: Complexity of valid trajectories under `gpt-4o` and `gpt-5.5` generators (planner, executor, refinement judge, and back-translation). The simulator, validity judge, query labeler, and dependency judge remain `gpt-4o`. F: unique functions; $|V|$: function calls; ℓ: longest dependency path; $|E|$: dependency edges.

B.7 Effect of a Stronger Generator Backbone

To examine whether our findings depend on the `gpt-4o` backbone, we rerun the full pipeline with the generator components (planner, executor, refinement judge, and back-translation) replaced by `gpt-5.5`. We keep the simulator, validity judge, query labeler, and dependency judge at `gpt-4o`, so that the generator is the only variable changed (table 8).

Two observations follow. First, the pipeline benefits from a stronger backbone rather than merely compensating for a weaker one: moving to `gpt-5.5` increases the complexity of BRIT on every metric ($|V|$ by 63% and $|E|$ by 100%). Second, a stronger backbone alone does not close the gap: the re-implemented single-call baselines remain nearly unchanged ($|V| \approx$ 1–2 under both backbones). The query-first ablation also benefits from `gpt-5.5` and slightly exceeds BRIT on dependency edges, indicating that the gains are attributable primarily to the pipeline design (planning, execution, and refinement) rather than to backbone capability.

C Additional Discussions (Continued)

Promising advances in synthetic tool-calling data generation. Recent synthetic function-calling approaches have targeted improvements at different pipeline stages. ToolMind (Yang et al., 2025b) enhances trajectory construction in the **generator** by modeling dependency relations between tools and performing random walks to generate sequences with data dependencies, which scaffold multi-turn conversation synthesis. ButtonInstruct (Chen et al., 2024) improves the **generator** through composition: it creates atomic query-execution pairs and composes them into complex intents requiring multiple calls, better reflecting real-world tool-calling complexity. Similarly, Arcadinho et al. (2024) improves the **generator** by converting procedural documents into flow-graphs, using the resulting operation sequences to synthesize complex tool-calling instances. Rather than modifying the **generator** module like these approaches, APIGen-MT (Prabhakar et al., 2025), a follow-up to APIGen (Liu et al., 2024b) only modifies the **post-processor** by incorporating feedback mechanisms to identify and correct errors, iteratively improving data quality—an approach aligned with our strategy.

It is encouraging to see recent efforts to improve synthetic data quality through refined generation and filtering techniques. However, existing approaches typically refine only a single stage of the pipeline. Our theoretical analysis with suitable probabilistic assumptions reveals a key insight: *jointly optimizing both the generator and post-processor is necessary to achieve a heavy-tailed complexity distribution*. This finding suggests that practitioners should adopt end-to-end strategies addressing both stages rather than isolated improvements, ultimately yielding higher-quality tool-calling datasets.

Finally, it is important to note that these works focus on multi-turn dialogue synthesis containing multiple successive user queries where each one is paired with single function call—a setting fundamentally different from our compositional single-query approach. This distributional difference precludes direct comparison with existing methods. Furthermore, the lack of publicly released code and intermediate artifacts (such as compositional intents and their corresponding execution traces that are used for constructing the final multi-turn conversations) makes systematic evaluation against these approaches infeasible and hence, not included in our comparisons.

Additional factors that influence difficulty. Beyond trajectory complexity, several other factors significantly influence tool-calling problem difficulty. τ-bench (Yao et al., 2024) demonstrates that domain-specific policy adherence and multi-turn user interactions introduce substantial challenges, as agents must follow guidelines while handling dynamic conversations beyond simple API calls. The BFCL (Charlie Cheng-Jie Ji, 2024) reveals that while models excel at single-turn calls, memory management, dynamic decision-making, and the ability to abstain from acting when tools should not be invoked remain significant challenges. ACEBench (Chen et al., 2025) further captures this diversity by distinguishing between normal scenarios, special cases with ambiguous or incomplete instructions, and agent-based interactions simulating real-world multi-turn dialogues. Additionally, factors such as function dependency depth, the presence of irrelevant functions with type-compatible variables, and the need for state tracking across multi-step sequences significantly impact model performance (Maekawa et al., 2025; Lu et al., 2025).

These observations suggest that effective data generation should not only target trajectory complexity but also incorporate diverse instruction clarity, harmful cases, agentic applications, domain policies, and function-schema related structures to create comprehensive and realistic tool-calling datasets. While there are independent works addressing some of these considerations, there remains a gap in systematically generating complex, compositional queries that require lengthy, multi-step solution trajectories. Our work aims to fill this gap by providing a principled approach to creating such challenging, realistic tool-calling datasets.

Practical validity of assumptions When generating highly complex datapoints, node validity may exhibit error propagation and compounding errors, where failures in earlier steps influence subsequent ones. Consequently, the probability of correct trajectory generation may decay faster than the independent node correctness assumption predicts. However, this may not necessarily invalidate theorem 3.1 and theorem 3.2. If the true validity probability decays faster than the assumed error model (see assumption 2.1) for sufficiently large c, the summands in the true validity distribution's moment generating function decays faster than that of our assumed distribution, preserving the theorems' conclusions.

While we assume iterative refinement resolves each error with fixed probability, some problem instances may remain unresolvable even with unlimited refinement budget. Such residual failures arise when models fail to comprehend problem requirements, misunderstand function arguments and schemas, or cannot extract relevant data from prior function calls needed for subsequent ones (Abdollahi et al., 2025). We urge future work to focus on more realistic assumptions grounded in empirical findings and perform similar tail analysis to derive useful theoretical insights.

Is our theoretical findings applicable for compositional problem solving? Given that solutions to compositional problems can be represented as computational graphs (section 2.2.2) and that we have employed the independent node correctness assumption (assumption 2.1), which is directly relevant for LLM-based compositional problem solving, a natural follow-up is to examine whether our results apply to compositional queries. To do so, we must first understand what characterizes the heavy tail for such queries. Empirically, word frequency in user queries follows a Zipfian distribution (Powers, 1998; Lillo & Ruggieri, 2021; Piantadosi, 2014). This suggests the heavy tail may reflect the distribution of search phrases rather than problem complexity. Consequently, our computational theory may not directly extend to such cases.

D TRAINING SETUP

All fine-tuning experiments were conducted on a single A100 (80GB) GPU using the LlamaFactory library with its default LoRA configuration. Each fine-tuning run completed within 4 hours.

LIMITATIONS

Refer appendix C for a more comprehensive discussion. Here, we summarize the key limitations of our work.

- **Theoretical assumptions**: Our theoretical analysis relies on an independent node error model that may not fully capture the interdependencies and cascading failures inherent in complex function-call sequences. Additionally, the error correction model may not hold universally across different generation settings.
- **Scope of complexity**: Trajectory complexity is defined solely by the number of function calls; there may be other important difficulty factors (ambiguous instructions, domain policies, memory management) that are equally important, but are beyond the scope of our work.
- **Limited baseline comparisons**: We are unable to directly compare against recent concurrent methods (ToolMind, ButtonInstruction, APIGen-MT) due to unavailable implementations or missing intermediate artifacts necessary for fair evaluation.