Quality Matters: Evaluating Synthetic Data for Tool-Using LLMs

Shadi Iskander*
Amazon
shadisk@amazon.com

Nachshon Cohen
Amazon
nachshon@amazon.com

Zohar Karnin
Technology Innovation Institute
zohar.karnin@tii.ae

Ori Shapira
OriginAI
obspp18@gmail.com

Sofia Tolmach
Amazon
sofiato@amazon.com

Abstract

Training large language models (LLMs) for external tool usage is a rapidly expanding field, with recent research focusing on generating synthetic data to address the shortage of available data. However, the absence of systematic data quality checks poses complications for properly training and testing models. To that end, we propose two approaches for assessing the reliability of data for training LLMs to use external tools. The first approach uses intuitive, human-defined correctness criteria. The second approach uses a model-driven assessment with in-context evaluation. We conduct a thorough evaluation of data quality on two popular benchmarks, followed by an extrinsic evaluation that showcases the impact of data quality on model performance. Our results demonstrate that models trained on high-quality data outperform those trained on unvalidated data, even when trained with a smaller quantity of data. These findings empirically support the significance of assessing and ensuring the reliability of training data for tool-using LLMs.

1 Introduction

Enabling LLMs to make use of external tools is a promising frontier that allows tapping into information that is not readily available to the model itself (Huang et al., 2024; Li et al., 2023a; Qin et al., 2024; Tang et al., 2023; Yang et al., 2023; Patil et al., 2023; Schick et al., 2023). Given a request and a list of available external API functions, the basic task of a model is to collect information by invoking functions, and then to generate a response for the request. Due to the lack of data for the task and the high cost of creating such data, researchers have devised synthetic datasets, predominantly with the assistance of LLMs (Huang et al., 2024; Li et al., 2023a; Tang et al., 2023). These datasets have facilitated a great leap in promoting the appealing applications of tool-using LLMs.

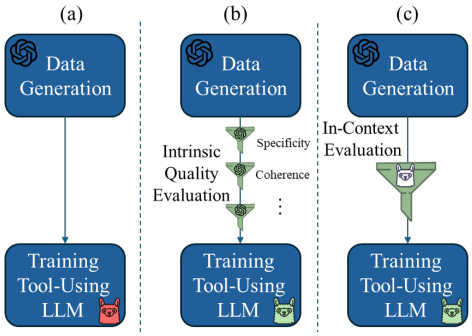


Figure 1: Data quality assessment methods for improving the training process of tool-using LLMs (a), employing two different approaches: (b) intrinsic quality evaluation, using an external LLM to measure various human-defined criteria; (c) in-context evaluation, using the target LLM to measure the educational value of data instances. A smaller high-quality training dataset is more effective than a larger unvalidated set.

Recently, Zhou et al. (2023) showed that higher quality training data yields better performance by LLMs in text generation tasks. However, leading works on tool-using LLMs have not made an effort to measure the quality of training data. Rather, only model outputs are extrinsically evaluated, disregarding the effect of the data on the tested models. Most research on tool-using LLMs focuses on improving training and evaluation processes (Huang et al., 2024; Qin et al., 2024; Tang et al., 2023). The lack of attention to *data quality* makes it difficult to interpret potential pitfalls for models. In turn, this wastes valuable resources for configuring and tuning models over possibly erroneous data.

Datasets for tool-using LLMs comprise instructions and ground truth API call sequences, and are created mainly with LLMs. Two such prominent datasets (Qin et al., 2024; Tang et al., 2023) were produced with the help of ChatGPT (OpenAI, 2024), and were not explicitly assessed for their quality. A closer inspection, conducted in this work,

*This work was done during an internship in Amazon and as part of graduate studies at the Technion - Israel Institute of Technology.

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 4958–4976
November 12-16, 2024

reveals numerous errors within the data, both in the instructions and in the ground-truth API calls (§4).

To conduct our inspections, we define intrinsic measures for data quality assessment, focusing on different aspects of quality. For each aspect we outline human evaluation guidelines, as well as implement automated methods for evaluation. The automatic methods employ ChatGPT, either by directly asking for its evaluation or by having it perform a proxy task and deriving the evaluation from its output. We show high agreement for our automated methods with expert human annotations. In addition to the intrinsic measures, we propose a metric we call *In-Context Evaluation* (ICE; §5). ICE evaluates a data instance by how helpful it is for in-context learning, thus predicting its helpfulness for training a model (§5). This metric is fully automated and does not rely on task-specific measurement definitions.

Other than being appraisal instruments, the intrinsic evaluation and ICE metrics can be used to automatically filter out low quality data from an existing dataset. In Section 6 we carry out this procedure, and display the effect of training tool-using LLMs with higher quality data. Our findings, demonstrated on the ToolBench (Qin et al., 2024) and ToolAlpaca (Tang et al., 2023) benchmarks, show either better or comparable performance when using a small high-quality training dataset, compared to the original models trained on larger unverified datasets. The two benchmarks are based on different API function sets, and different data generation and training methods, indicating the generalized applicability of our methods.

2 Background and Related Work

2.1 External Tool Usage by LLMs

Tool learning is a recent area of research, aiming to enable LLMs to overcome limitations by accessing tools for, e.g., retrieving up-to-date information (Kasai et al., 2023; Cheng et al., 2024), or performing mathematical calculations (Schick et al., 2023), thereby enhancing their usability for real-world needs.

Research on tool learning focuses on various aspects of training LLMs to use external tools. These mainly include tool selection, tool usage, and planning (Zhuang et al., 2023; Qin et al., 2024; Patil et al., 2023; Yao et al., 2023). Such models are mainly evaluated extrinsically, only measuring the final results. T-Eval (Chen et al., 2024) is the first evaluation framework that analyzes tool-using LLMs intrinsically. That is, it decomposes the evaluation into all sub-tasks (such as selection, usage and planning), measuring the fine-grained abilities of models as tool agents. We intrinsically evaluate the *data* for tool-usage instead of a *model*.

2.2 Data Generation

Recent notable works generated synthetic data for tool learning. ToolBench (Qin et al., 2024) leverages a large pool of real API functions.[1] ChatGPT (OpenAI, 2024) was used to generate an instruction that would require invoking a given small set of these tools, as well as to produce a solution path for the respective instruction. The data was constructed with a varying number of tools per instance and varying relatedness between the tools. API Bank (Li et al., 2023a) created synthetic API documentation, instruction queries, and responses using strong LLMs (GPT-4 and ChatGPT; OpenAI, 2024). A smaller test set was created and validated manually by humans. Tang et al. (2023) constructed the ToolAlpaca dataset using ChatGPT to generate cleaner documentation upon existing APIs, and respective instructions and responses. In ToolAlpaca, most synthesized instructions only require a single tool to fulfill the request. The test set was validated by humans to ensure quality.

To strengthen the credibility of our findings in this work, we conduct our experiments over both ToolBench and ToolAlpaca, which differ in API quality and instruction requirements.

2.3 Data Quality

The ever-increasing dependence on data for training large models has paved a line of work that analyzes the effect of data quality on fine-tuning models. Findings show that a small but high-quality dataset can be highly effective for fine-tuning a relatively small model, surpassing the performance of a larger model. For example, Phi (Gunasekar et al., 2023; Li et al., 2023b) explored code generation tasks and prompted GPT-4 to assess the educational value of coding examples. They demonstrated that a small number of high-quality and diverse examples are sufficient to reach good quality of code generation. In the realm of instruction tuning, Li et al. (2024) suggest employing self-augmentation and self-curation to iteratively improve the set of instructions used for instruct-tuning an LLM. LIMA

[1] Based on RapidAPI: https://rapidapi.com/hub

(Zhou et al., 2023) considers the broader picture of data quality, and show that as few as 1000 high-quality examples can be sufficient for training an instruction-following model.

Our work differs from these studies in that it applies to the regime of tool usage. It can be seen as additional evidence reinforcing the prevailing "less is more" trend, proving the importance of data quality in this regime.

3 Task Setup

Tool-using LLMs are expected to behave as follows. Given a set of tools $T = \{t_1, ..., t_n\}$, represented as API functions, and an instruction query q, a model is required to plan a call sequence $S = (t'_1, ..., t'_k)$, based on T, that would obtain information, or perform actions, needed to address q. Based on the responses obtained after performing the call sequence (using an external API invoker), the model then generates a final response r that responds to q. The primary method for model evaluation is based on calculating the *pass rate*, which measures the proportion of instances that successfully addressed their instructions, i.e., a predicted r responded to q adequately (explained further in §6).

As mentioned in Section 2, the prominent datasets created for training and testing tool-using models were created synthetically with the assistance of LLMs. Specifically, we utilize the ToolBench (Qin et al., 2024) and ToolAlpaca (Tang et al., 2023) datasets. Table 1 summarizes their characteristics. The main practical differences are the quality of the APIs (i.e., the documentation clarity and uniformity of ToolBench is inferior to that of ToolAlpaca), and the number of tools required to respond to a query instruction (ToolBench might require several calls to unrelated tools, while ToolAlpaca requires calling a maximum of two related tools). As presented later in this work, these two differences strongly reflect on the overall quality of the respective datasets.

Characteristic	ToolBench	ToolAlpaca
API source	real-world	synthesized w/GPT
# available APIs	16K	2.3K
# of training instances	125K	4.2K
# required API calls per instance	1-5	1-2

Table 1: Summary of relevant dataset characteristics.

Problem statement. Our primary focus is on evaluating and improving data quality, and to show its effect on model performance in tool-using LLMs. Following a similar line of research, we hypothesize that a small quantity of high-quality training data is preferred over a large quantity of lower-quality data. To demonstrate this, we first define intrinsic quality criteria for the data (§4.1) and implement automated metrics accordingly (§4.3). We additionally propose an alternative data quality appraisal method using in-context evaluation (§5). Finally, we filter out the lower-quality data from datasets using our automated metrics, and analyze the effect of the improved data quality on model performance (§6).

4 Intrinsic Quality Evaluation

4.1 Quality Criteria

We set out to understand what makes an instance of data high quality, specifically for training tool-using LLMs. The criteria we discuss pertain to both the query instruction and the API call sequence of a data instance.[2]

4.1.1 Instruction Properties

In our setting, an instruction is a free-form text of one-to-a-few sentences that describes a user requirement. An instruction can contain more than one request, likely implying the need for several tool invocations. The following properties in the instruction demand validation (examples in Table 2):

Specificity. All the required details are present in the instruction for the LLM to be able to fulfill the user requests[3].

Coherence. The requests within the instruction are logically related, and the order of requests makes sense for a real-world use case.

Solvability. The requests within the instruction can be addressed by the given API tools.

4.1.2 API-Call Sequence Properties

Apart from the instruction, given as input to a model, the other vital component of a training instance is the ground-truth output used for training (or evaluating) a model. In our setting, this is the sequence of API calls that the model is expected to infer. We define the following properties for API-call sequence correctness (see Table 3 for examples):

[2]We considered other properties that were eventually excluded from our framework, such as diversity and syntax validity. See Appendix A.1 for more details.

[3]Note that we deal with a setting where the agent is expected to complete the instruction without asking clarification questions.

Synthetic Instruction	Error Type
I'm curious about **a famous actor's** career. Can you provide details about their filmography, including their best-known titles and streaming availability on Netflix, Hulu, and Prime Video? Also, share some interesting facts about the actor.	Low Specificity
As a language enthusiast, I'm always eager to learn new languages. Can you help me explore the possible translations between Russian, Japanese, and Arabic? **Additionally, I would like to obtain a list of available language codes for future reference**.	Low Coherence
I need to **create** a temporary email address with the domain 'example.com'. Once created, I want to fetch the latest message from this email address. Given APIs: [Get list of domains for email, Get message by message ID]	Unsolvable

Table 2: Examples of synthesized instructions, **highlighted** with errors involving our defined properties.

Synthetic Instruction	API-Call within Sequence	Error Type
Can you create a shield logo for my friend's blog? The name of the blog is 'The Creative Mind'.	`generate_shield(name=None)`	Missing Parameter
I need to fetch the current weather conditions for a specific location. Can you help me by providing the address and geocoordinates of the location?	`geocode(address="San Francisco")` ...	Hallucinated Parameter

Table 3: Examples of synthesized API-call sequences for respective instructions, with incorrect parameters.

Parameter alignment. The parameter values in each of the API calls are correctly extracted or inferred from the instruction, there are no missing or hallucinated parameter values.

Sufficiency. The API-call sequence applies to all required actions for the instruction's requests.

Minimality. The API-call sequence would address all the instruction requirements with a minimal number of API calls. No unnecessary or redundant API calls are included in the sequence.

4.2 Manual Annotations

The six intrinsic properties defined above specify the desired qualities for data instances of tool-using LLMs. Existing datasets do not always abide by these quality criteria, especially when they are collected synthetically and do not go through a cleaning phase. We inspect such noisy data by preparing annotation guidelines with respect to the criteria, and annotating accordingly. Specifically, we methodically[4] annotated 50 (instruction, API sequence) pairs from each of the training sets of ToolBench (Qin et al., 2024) and ToolAlpaca (Tang et al., 2023), as well as a large portion of the ToolBench test set (∼700 instances).[5] Each of the criteria is marked either as valid or invalid for each of the annotated instances. The annotated data is used in later sections for analyses and experiments.

4.3 Automated Metrics

Although manual assessment of data is preferred for its reliability, it is labor-intensive and therefore not scalable or practical. We propose automatic metrics for the intrinsic quality criteria defined above. The metrics are based on ChatGPT,[6] which is tasked to determine the validity of each criterion as a binary decision.

For the dimensions of Specificity, Coherence and Parameter alignment, direct annotation with ChatGPT proved to be challenging. That is, simply asking the model to validate the property in a natural language instruction did not yield sufficient decisions (see Appendix A.3). Thus, we transformed the direct annotation tasks into traditional NLP tasks, on which ChatGPT performed better.

Specificity. Validating the specificity of requests is modeled as an *extraction* task. ChatGPT is tasked to infer the details required for a given request, and then extract the available values from the instruction, or mark a parameter as `#missing`. We then compute a proxy score for specificity: 1 if all parameters were successfully extracted from the instruction, and 0 otherwise[7].

[4] Annotation process and agreement in Appendix A.4.

[5] We did not review ToolAlpaca's test set since it is already manually verified.

[6] Throughout the paper, we use `gpt-3.5-turbo-0613`.

[7] A continuous score can be computed as the percentage of extracted parameters from the total number of parameters, but we opted for a binary score for simplicity.

		ToolBench Dataset				ToolAlpaca Dataset			
	Quality Criterion	**Accuracy**	**Prec.**	**Rec.**	**F1**	**Accuracy**	**Prec.**	**Rec.**	**F1**
Instruction	Specificity	0.74	0.70	0.84	0.76	0.88	0.75	0.86	0.80
	Coherence	0.82	0.62	0.77	0.69	0.98	0.50	1.00	0.66
	Solvability	0.90	0.70	0.78	0.74	0.92	0.75	0.50	0.60
	Instruction Correctness	0.72	0.72	0.90	0.80	0.86	0.80	0.84	0.82
API Call Seq.	Parameter Alignment	0.70	0.63	0.92	0.74	0.76	0.74	0.80	0.77
	Sufficiency	0.78	0.64	0.60	0.62	0.88	0.80	0.50	0.62
	Minimality	0.76	0.95	0.63	0.76	0.86	0.88	0.57	0.70
	Sequence Correctness	0.82	0.83	0.94	0.88	0.76	0.70	0.85	0.80
	Overall Correctness	0.86	0.89	0.95	0.92	0.76	0.74	0.90	0.81

Table 4: Validation results of the automated metrics for each criterion compared against human annotations. Coarse-grained correctness considers combined correctness over specific criteria. Note that precision, recall and F1 are measured w.r.t. a label that is positive when an error occurs, so e.g., recall means the amount of errors caught.

Coherence. We adopt the concept of *next sentence prediction* to assess coherence. The instruction is split into sentences, and ChatGPT determines if each subsequent sentence logically follows the previous one. We set a coherence score as 1 if all sentence pairs are judged logically connected, and 0 otherwise.

Parameter alignment. ChatGPT first extracts parameters (as in specificity), and then compares it to the ground truth parameter values.

Solvability, Sufficiency & Minimality. The remaining criteria use direct instructions to ChatGPT. The prompts used are provided in Appendix A.2.

4.3.1 Evaluation of Automated Metrics

Using the manually annotated data (described in §4.2), we conduct an assessment of the automatic metrics proposed. For each of the ToolBench and ToolAlpaca datasets, the 50 annotated instances are compared against the automatically produced values, producing measures of accuracy (agreement), precision, recall and F1 score. We treat instances marked as incorrect instances as positive labels, since we aim to identify and filter erroneous instances.

We conduct a coarser-grained evaluation of the criteria, assessing **Instruction Correctness** as incorrect if any instruction criterion is wrong, and **Sequence Correctness** as incorrect if any API-call sequence criterion is wrong. **Overall Correctness** aggregates all six criteria similarly.

Results are presented in Table 4. Given that our main objective is to identify and filter out incorrect data samples, our emphasis is on achieving high recall. This objective is largely met across most criteria in both datasets. In the Overall Correctness assessment, which aggregates all criteria, we observe high recall and precision, demonstrating a strong alignment of the automated metrics with human judgment. This approach thus offers a reliable mechanism to identify problematic data instances.

4.3.2 Quality of Datasets

Table 5 presents the percentage of instances containing errors in the train sets of both ToolBench and ToolAlpaca, as determined by the automated metrics. These statistics provide insights into the quality of the data in each dataset. In the ToolBench dataset we observe a much higher percentage of errors across most quality criteria, when compared to ToolAlpaca. This difference may be attributed to (1) the complexity of instructions in ToolBench, which can require several (up to 5) API calls; (2) real-world APIs used in ToolBench, where the API documentation is not always clear, resulting in incorrectly generated instructions and API-call sequences. Notice that in both datasets, over 33% of instances have parameter alignment errors. Such an error means that one of the core requirements of a tool-using model – identifying parameters correctly – is misleadingly learned in more than a third of the cases, due to wrong training examples. Some anecdotal examples of incorrect instructions found by our metrics can be seen in Appendix A.5.

We further explore the relationship between quality criteria within the datasets in Appendix A.6.

Dataset	Instruction				API-Call Sequence				Inst. & Seq. Overall
	Specificity	Coherence	Solvable	Overall	Param. Alignment	Sufficiency	Minimality	Overall	
ToolBench	20.4%	22.1%	18.2%	47.3%	47.9%	33.6%	45.1%	74.4%	84.0%
ToolAlpaca	17.5%	4.1%	12.7%	27.2%	33.1%	13.6%	15.9%	35.5%	44.8%

Table 5: Percentage of instances containing errors in each dimension, according to our automated methods, in the train sets of the examined datasets. This analysis is done on 125K examples in ToolBench and 4.2K in ToolAlpaca.

5 In-Context Evaluation (ICE) as an Alternative Data Measurement

Using intrinsic evaluation, we have defined an intuitive and straightforward approach to identify low-quality data instances based on human understanding of data correctness. However, assessing the "educational" value of an instance, i.e., its contribution to the learning process of a model, is a complex task. In addition, the intrinsic evaluation metrics proposed rely on prompting a powerful LLM, which can become costly on large datasets. To address these challenges, we propose In-Context Evaluation (ICE) as an alternative automatic approach for assessing data quality.

Recent studies found a connection between in-context learning and fine-tuning, demonstrating that language models implicitly perform gradient descent when dealing with in-context tasks (Von Oswald et al., 2023; Dai et al., 2023). Motivated by this insight, we seek to evaluate the educational value of each data instance by measuring the performance of in-context learning using the specific instance as a one-shot example.

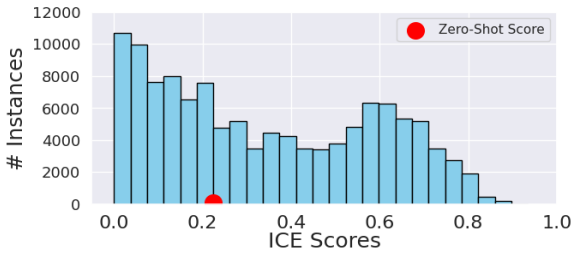
5.1 Setup

To construct the in-context task for external tool use, we prepare a set of 10 human-written APIs, denoted by A, with simple accompanying documentation. In addition, we hand-craft a set of 7 test query instructions, TEST, where each such example contains a natural language instruction and an expected API-call sequence, from the APIs in A, that would address the instruction. For each evaluation instance, we insert an in-context example, x, which consists of an instruction and API-call sequence from the training dataset (i.e., ToolBench or ToolAlpaca). x follows the structure of the test examples. We then formulate a prompt for the LLM that we aim to train, that asks to generate responses for the 7 test cases. In particular, the prompt includes (1) task instructions, (2) the API documentation of A, (3) the training instance, x, given as a one-shot example, (4) the 7 testing instructions of TEST.

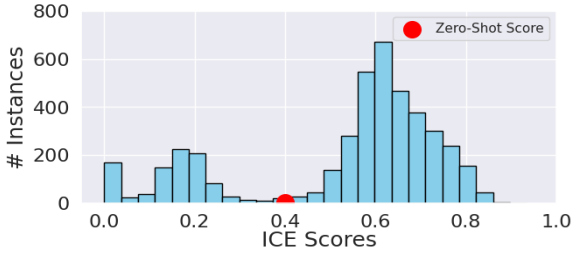
The prompt is given to an LLM we aim to train: LLaMA-7B for ToolBench or Vicuna-7B for ToolAlpaca. We analyze its response, that should include the 7 API-call sequences of TEST. The responses for the test instructions are evaluated against the ground truth (using Levenshtein similarity (Levenshtein et al., 1966), expecting an exact match for API-call sequences). The final ICE score for x is the average over the 7 test examples, interpreted as a measure of the educational value of x. We provide the full prompt and the precise way we compute the ICE score in Appendix B.

5.2 Analysis

Score distribution. We present ICE scores for both datasets in Figure 2. Interestingly, the ICE scores distribution in ToolAlpaca exhibits bimodal distribution, which suggests the presence of two types of examples: one with higher ICE scores, which we expect to correlate with good-quality examples, and another with lower ICE scores, which is expected to lean towards low-quality



(a) ToolBench



(b) ToolAlpaca

Figure 2: Distribution of ICE scores. Most instances in ToolAlpaca are beneficial as the one-shot in-context example. ToolBench instances are not as effective.

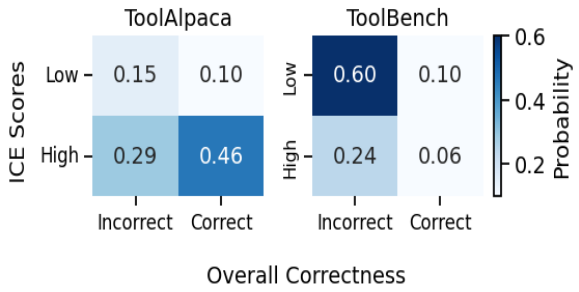


Figure 3: Confusion matrices comparing ICE scores and human Overall Correctness scores.

examples. The majority of instances in ToolAlpaca have relatively high ICE scores – indicating high overall dataset quality. In contrast, most samples in ToolBench have low ICE scores, suggesting that the overall data quality in this dataset may be lower compared to ToolAlpaca. This observation is consistent with the analysis presented using the intrinsic evaluation in Section 4.3.2.

Correlation to human-defined criteria. ICE is a model-driven assessment method that may not necessarily align with human-defined correctness criteria. To investigate the relationship between ICE approach and human-defined criteria, we divide the datasets into low and high ICE scores using a threshold of 0.5. We then generate confusion matrices between ICE scores and human *Overall Correctness* scores. As seen in Figure 3, ICE scores correlate with human-defined correctness to some extent, showing it is a sensible metric and can be beneficial as an alternative method for filtering data. On the other hand, this correlation is far from perfect, showing that ICE is inherently different from human-prescribed correctness. In Section 6 we test ICE both as an alternative and as a complementary filtering technique to human-defined correctness.

6 Extrinsic Evaluation

In this section, we validate our main claim that fine-tuning a tool-using LLM with a smaller dataset of high-quality data can lead to better performance of the model on the task, compared to a larger noisy dataset. We use both intrinsic metrics and ICE to create training sets of varying quality, and compare the results of training with the different sets.

Training setup. We follow the general setup used by the ToolAlpaca (Tang et al., 2023) and ToolBench (Qin et al., 2024) benchmarks. Specifically, we fine-tune Vicuna-7B (Chiang et al.) for ToolAlpaca, and LLaMA-7B (Touvron et al., 2023) for ToolBench, both using LoRA (Hu et al., 2022) (see Appendix C.1 for more details).

We use the following train sub-sets from each model's respective benchmark training sets:

- **Random Sample**: uniform random subset.
- **High Instruction**: uniform sample of instances with all three instruction criteria intact.
- **High Instruction + Seq**: uniform sample of instances with all six criteria intact.
- **Low ICE**: instances with the lowest ICE scores.
- **High ICE**: instances with the highest ICE scores.
- **High Instruction + Seq + ICE**: instances with all six criteria intact and high ICE scores.
- **Original**: the full original training set.

Each fine-tuned model is evaluated using **pass rate**, which is an extrinsic evaluation procedure used in both benchmarks.[8] This measures the proportion of instances in which the resulting API-call sequences and responses adequately address their respective instruction query. See Appendix C.2 for more details on the evaluation procedure.

Test sets. For ToolAlpaca we use the original test set, as it is created with human annotation. It consists of 100 instructions of simulated tools that were not part of the training tool set. ToolBench test set was created using LLMs and was not manually validated. We inspected 674 examples, as detailed in Appendix A.4. For instances of low quality, we either rectified them (e.g., manually adding a missing parameter value), or discarded them. The resulting test set contains 420 high-quality examples.[9]

6.1 Main Results

Results are presented in Table 6, where the training sub-sets are fixed to size 10K for ToolBench and 2K for ToolAlpaca. The results demonstrate the impact of training data quality on model performance.

When comparing to a model fine-tuned on a random subset of the original training data (row 1), all methods of filtering low-quality instances (rows 3-6) are clearly beneficial. Moreover, when fine-tuning models with *much smaller* high-quality sub-sets (rows 3-6), performance is comparable or superior to models fine-tuned on the *full* original

[8] In ToolAlpaca this metric is referred to as "overall accuracy", although it conveys the same concept.

[9] This test set is available in the supplementary material.

	Fine-tune Set	ToolBench			ToolAlpaca		
		Size	Pass Rate	95% CI	Size	Pass Rate	95% CI
1	Random Sample	10K	0.35	(0.31, 0.39)	2K	0.48	(0.38, 0.58)
2	Low ICE	10K	0.24	(0.20, 0.28)	2K	0.48	(0.38, 0.58)
3	High ICE	10K	0.43	(0.38, 0.47)	2K	0.54	(0.44, 0.64)
4	High Instruction	10K	0.49	(0.44, 0.53)	2K	0.52	(0.42, 0.62)
5	High Instruction + Seq	10K	0.52	(0.47, 0.56)	2K	0.54	(0.44, 0.64)
6	High Instruction + Seq + ICE	10K	0.54	(0.49, 0.58)	2K	0.55	(0.45, 0.65)
7	Original	73K†	0.45	(0.40, 0.49)	4.2K	0.56	(0.46, 0.66)

Table 6: Extrinsic evaluation results with confidence intervals, and the size of the training sets. By filtering out low-quality training instances, the models perform significantly better than (in ToolBench) or as good as (in ToolAlpaca) the original models that use a much larger unvalidated training set. † Although there are 125K instances in the released dataset, the model published in the original paper was trained on a subset of 73K instances.

training sets (row 7). Consistent with the findings on the ToolBench dataset's lower overall quality (§4 and §5), results indicate improved model performance with a high-quality subset, comprising only ∼14% of the original dataset's size (row 6).

Comparing the intrinsic metrics to the ICE method, we find that the former is a better mechanism for filtering training data (row 3 vs. 5). Using both techniques together can be marginally better (row 5 vs. 6). Another insight to consider is that taking data with low ICE scores (row 2) is indeed harmful to model performance, further reinforcing that the method is valuable despite its partial agreement with intrinsic human-defined criteria (§5).

In ToolAlpaca, the gaps are less pronounced than in ToolBench, likely influenced by: (1) the higher quality of the original dataset, (2) smaller original training set, causing the filtered datasets to be too small, (3) smaller test set, only 100 instances. Nonetheless, the trend still exists (albeit being within the confidence intervals). This, combined with the intrinsic assessment of Table 5, provides encouraging evidence for the effectiveness of our methods, even for this smaller-scale dataset.

6.2 Data Scaling Analysis

To further explore the effects of training tool-using LLMs with high-quality data, we analyze the performance of models when fine-tuning with *different sizes* of train sets. We focus here on ToolBench, where the impact is more significant and the original training set is larger, and use subsets with sizes ranging from 1K to 20K for the different filtration methods. Results can be found in Figure 4. As size increases, we observe consistently better performance, with an expected plateau in the largest dataset sizes. Notice that at some point the training datasets have no more high-quality data instances that pass our filters, putting a natural limit on our experiments.

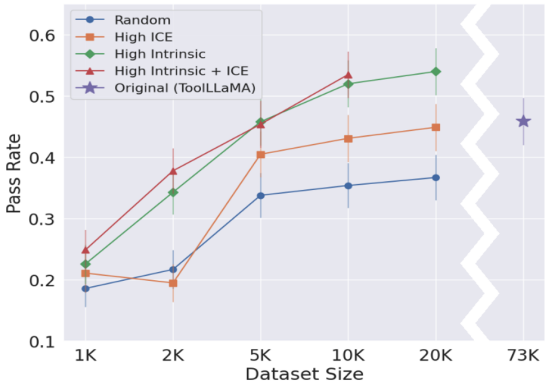


Figure 4: Pass rate results in ToolBench when using train sets with different sizes and filtration methods.

7 Conclusion

We demonstrated the importance of evaluating the quality of training data for fine-tuning tool-using LLMs. We introduce two data-evaluation approaches. The first is a rigorously devised intrinsic quality assessment, for which we implement automated metrics. The second uses in-context evaluation, that measures the educational value of training examples. While the former method is more explainable and dependable, the latter is computationally cheaper. We apply both

approaches to filter data instances from two large datasets of differing qualities. The resulting subsets of training data demonstrate comparable or superior quality in terms of model performance, despite their smaller size compared to the original datasets. Overall, we observe that it is worthwhile to more carefully choose the training data for tool-using LLMs. If investing in better methods of data generation is costly, automatic post-hoc filtration can be a great alternative.

8 Limitations

In this work, we address the quality of data instances, and refrain from overall dataset-level quality criteria, primarily diversity of data. Our focus is on instance-level quality, and we show the advantage of training LLMs with data that is identified as high-quality with instance-level criteria only. Future work can explore the benefits of dataset-level quality criteria as well.

Our experiments span over two popular benchmarks for tool-using LLMs. They are differing in characteristics and quality, and can therefore provide insights that are not benchmark-specific. Nevertheless, conducting our analyses on additional related datasets and LLMs would provide an even more generalized representation of our results.

References

Asli Celikyilmaz, Elizabeth Clark, and Jianfeng Gao. 2021. Evaluation of Text Generation: A Survey.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023. Extending Context Window of Large Language Models via Positional Interpolation.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. 2024. T-Eval: Evaluating the Tool Utilization Capability of Large Language Models Step by Step.

Jeffrey Cheng, Marc Marone, Orion Weller, Dawn Lawrie, Daniel Khashabi, and Benjamin Van Durme. 2024. Dated Data: Tracing Knowledge Cutoffs in Large Language Models.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90% ChatGPT Quality. https://lmsys.org/blog/2023-03-30-vicuna/. Accessed: 2024-04-01.

Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Shuming Ma, Zhifang Sui, and Furu Wei. 2023. Why Can GPT Learn In-Context? Language Models Secretly Perform Gradient Descent as Meta-Optimizers. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 4005–4019, Toronto, Canada. Association for Computational Linguistics.

Zhiqiang Gong, Ping Zhong, and Weidong Hu. 2019. Diversity in Machine Learning. *IEEE Access*, 7:64323–64350.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. 2023. Textbooks Are All You Need.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, and Lichao Sun. 2024. MetaTool Benchmark: Deciding Whether to Use Tools and Which to Use. In *The Twelfth International Conference on Learning Representations*.

Jungo Kasai, Keisuke Sakaguchi, yoichi takahashi, Ronan Le Bras, Akari Asai, Xinyan Velocity Yu, Dragomir Radev, Noah A. Smith, Yejin Choi, and Kentaro Inui. 2023. RealTime QA: What's the Answer Right Now? In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Vladimir I Levenshtein et al. 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710. Soviet Union.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023a. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore. Association for Computational Linguistics.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Omer Levy, Luke Zettlemoyer, Jason E Weston, and Mike Lewis. 2024. Self-Alignment with Instruction Backtranslation. In *The Twelfth International Conference on Learning Representations*.

Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. 2023b. Textbooks Are All You Need II: phi-1.5 technical report.

OpenAI. 2024. GPT-4 Technical Report.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large Language Model Connected with Massive APIs.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551. Curran Associates, Inc.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. ToolAlpaca: Generalized Tool Learning for Language Models with 3000 Simulated Cases.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models.

Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, Joao Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. 2023. Transformers Learn In-Context by Gradient Descent. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 35151–35174. PMLR.

Hui Yang, Sifu Yue, and Yunzhong He. 2023. Auto-GPT for Online Decision Making: Benchmarks and Additional Opinions.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*.

Yue Yu, Yuchen Zhuang, Jieyu Zhang, Yu Meng, Alexander J Ratner, Ranjay Krishna, Jiaming Shen, and Chao Zhang. 2023. Large Language Model as Attributed Training Data Generator: A Tale of Diversity and Bias. In *Advances in Neural Information Processing Systems*, volume 36, pages 55734–55784. Curran Associates, Inc.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, LILI YU, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. 2023. LIMA: Less Is More for Alignment. In *Advances in Neural Information Processing Systems*, volume 36, pages 55006–55021. Curran Associates, Inc.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. ToolQA: A Dataset for LLM Question Answering with External Tools. In *Advances in Neural Information Processing Systems*, volume 36, pages 50117–50143. Curran Associates, Inc.

A Intrinsic Evaluation

A.1 Other Quality Criteria

We outline here three quality criteria that are commonly addressed in the domain of data quality evaluation, and that we did not include in this work. (1) **Fluency** is the lexical quality of the text in terms of grammar, spelling, and style (Celikyilmaz et al., 2021). The reason for omitting this dimension is that the lexical quality of texts generated by powerful LLMs is very high. We found that virtually all instances in an assessment set had highly fluent texts. (2) **Syntax Validity** is whether the function calls and parameter names (not values) in the API-call sequence are valid. Using both manual validation and automatic rule-based lexical matching we found that data generated with ChatGPT did not exhibit such errors. (3) **Diversity** captures how different the data instances are amongst themselves in terms of assortment of requests, tool usage, difficulty, length and other properties. Similarly to other tasks and domains, it is expected that an LLM would learn to generalize better given diverse examples (Gong et al., 2019; Yu et al., 2023). We focus on instance-level criteria, and leave dataset-level criteria, such as diversity, for future work.

A.2 Prompts for Assessment

In Figures 6–10 we provide the prompts we use for automatically assessing the six human-defined quality criteria, using ChatGPT.

A.3 Unsuccessful Prompts

Direct questioning and annotation instruction with ChatGPT did not work well for the criteria of *Specificity*, *Coherence* and *Parameter Alignment*. In Figures 11–13 we provide the prompts. In Table 7 we provide validation results of alignment with human annotation on the same subset of examples of the ToolBench dataset.

Criterion	Acc.	Precision	Recall	F1
Specificity	0.54	0.56	0.36	0.43
Coherence	0.74	0.50	0.46	0.48
Alignment	0.66	0.69	0.71	0.70

Table 7: Validation results when using the direct questioning approach for *Specificity*, *Coherence* and *Parameter Alignment*. Compare to Table 4, which shows higher scores for the prompts ultimately used.

A.4 Manual Annotation Process

A.4.1 Annotating Training Data

To initiate the annotation process, we examined the data and identified the quality criteria (as outlined in §4.1). We then went through several cycles of examination and refinement of respective guidelines.

An instance of annotation shows the instruction, the available API functions, and the API-call sequence that should solve the instruction. The annotator needs to mark level of specificity of the instruction (1 to 3), its coherence (1 to 3), whether it is solvable with respect to the available API functions (yes/no), the sequence call validity in terms of function availability (yes/no), parameter alignment in the calls (yes/no), whether the sequence call solves the instruction (yes/no), and whether it does so minimally (yes/no). See Tables 8, 9 and 10 for annotation instructions of the first three criteria.

The annotators (authors of this paper) first annotated the same 20 instances from ToolBench and discussed differences, culminating in strong agreement between the annotators. The averaged Kappa statistics for the first three criteria are: Specificity 0.674 ("substantial"), Coherence 0.508 ("moderate"), and Solvability 0.414 ("moderate"). Annotators were then assigned different samples of data, for a total of 50 instances from the ToolBench train set, and 50 from the ToolAlpaca train set. We used this data to assess the intrinsic metrics that we developed (§4.3).

A.4.2 Annotating the ToolBench Test Set

In comparison to annotation of training instances, the test set annotation differs in two major aspects. First, the test set does not include API-call sequences, but rather only the input instructions. A training instance consists of an API-call sequence in order to teach an LLM how to devise a solution for attaining a final result. However during test time, tool-assisted LLMs are typically evaluated on the final result, and not on the API-call sequence used to achieve the result. Second, in our cleaned test set, we do not only mark inadequate instances, but we also attempt to fix instructions so that they become usable. The ToolBench test set contains 1100 instances (distinct from the 125K instances), and only filtering out faulty instances would leave very few suitable ones. Essentially, we use the ToolBench test set as data to build upon instead of creating new data altogether, which would be a much costlier procedure. The ultimate goal is to

produce a high-quality test set of solvable multi-request instructions.

An instruction can fail on either specificity, coherence or solvability. Therefore, to repair an instruction we focused on the failing criteria and rewrote the instruction to mend the faults. We allowed for some creativity as long as the quality criteria were intact, and the same number of requests was kept within the instruction.

For example, *"I'm planning a family movie night and I want to watch some classic films. Can you suggest some iconic movies available on YouTube? Also, find a YouTube playlist of movie soundtracks. Additionally, provide the latest versions of C++, Objective-C, and Scala programming languages for my cousin who is a software developer."* Here, the first request ("suggest iconic movies") and the second request ("find a YouTube playlist") are not specific enough for the available API functions, and the third request ("provide the latest versions of C++...") is not coherent with the beginning of the instruction. We therefore rewrote the instruction for this instance as *"I'm learning how to program and I'd like some assistance. Can you suggest some videos on YouTube about C++? Also, download the video to MP3 from 'www.youtube.com/?123abc'. Additionally, please let me know the the latest versions of C++, Objective-C, and Scala programming languages."* The new instruction resolves the three issues described. In a case where it is unclear how to use the respective available API functions, no fix is made and the instance is simply discarded.

Five annotators annotated 674 of the 1100 instances in the ToolBench test set. 27.6% of the instances lacked specificity, 21.5% lacked coherence, and 32.7% were unsolvable. Overall, 37.7% of the instances were discarded, in cases where errors were too severe to be readily fixable. The new test set is used for measuring the performance of tool-using models in the multi-request setting (§6), and can generally be used as a high-quality benchmark. We provide the new test set in the supplementary material.

A.5 Qualitative Examples

In Tables 11 and 12 we provide examples of instructions which our method found as lacking specificity and coherence, from both ToolBench and ToolAlpaca datasets.

Specificity
Evaluate the extent to which the data examples contain all necessary information without gaps or missing variables for the AI assistant to address the user requests.
1 (Poor): The instruction is extremely broad and general, lacking essential information.
2 (Medium): The instruction includes moderate specific details but there are some gaps in information.
3 (Excellent): The instruction is highly specific and complete, with no significant missing information.

Table 8: Human annotation guidelines for Specificity.

Coherence
Evaluate the extent to which the different requests in the instruction are logically connected and relevant to each other.
1 (Poor): The different requests of the instruction are highly disjointed, lacking a logical connection.
2 (Medium): The different requests of the instruction have a moderate level of coherence but still possess some degree of separation.
3 (Excellent): The components of the instruction are highly coherent, with a strong logical connection.
Not Applicable: When there is only one request. (Considered as '3' for filtering.)

Table 9: Human annotation guidelines for Coherence.

Solvability
Determine if the ground truth APIs can handle the instruction in terms of functionality. It is alright if a parameter value is not explicitly provided in the query.
0 (No): The request cannot be handled by the given APIs. The APIs' functionalities do not fit or address the request.
1 (Yes): The instruction can be handled by using the given APIs. A parameter value might not be explicitly provided in the query.

Table 10: Human annotation guidelines for Solvability.

	Instruction Examples
From ToolBench	I'm planning to buy a used car and I need to decode the VIN number of a specific vehicle. Can you provide me with the car model, maker, year, engine, and other relevant information? Additionally, I'm curious about the trending search results on Google.
	I'm a wedding planner and I want to create personalized videos for my clients. Can you give me the details of a specific template I have in mind, including the variables it offers? Also, I need to access all my campaigns' information, including the images, videos, and image+video campaigns.
	I'm hosting a garden party next weekend. Can you give me the 1-hour/minutely forecast for the party location? Additionally, recommend some outdoor games and decorations for the event.
	I recently discovered a new song that I really love. Can you provide me with the lyrics and related data for the song? Also, suggest some similar songs that I might enjoy.
	I'm planning a trip to Europe and I want to stay updated on the energy prices in the region. Can you fetch all the available articles from a specific region, like Europe? Additionally, provide me with a list of news sources and their corresponding regions.
From ToolAlpaca	Please generate an invoice for my freelance work and send it to my client.
	How can I find the best gear for my character in Guild Wars 2?
	Hey, I'm planning a road trip and I want to check for any road closures along my route. Can you help me with that?
	I need to retrieve detailed information about a specific malware sample. Can you show me how to do that?
	I want to know if any of the email addresses in a list are disposable. Can you use the API to check which email addresses in the list are disposable?

Table 11: Examples of instructions which our method found as lacking **specificity**, from the two examined datasets.

A.6 Relationship Between Quality Criteria

We additionally explored the relationship between quality criteria within the datasets. Generally, the correlations between dimensions are not particularly high. A notable analysis we conducted shows the effect of *Specificity* on *Parameter Alignment*. As illustrated in Figure 5, when specificity is weak, it is also more likely that parameter alignment is weak. This might be expected behavior since low specificity means that parameter values are missing in the instruction, and the LLM hallucinates a value in order to complete its task. The correlation however is not exceedingly high, in particular we see in ToolBench that even for instances with high specificity, the parameter alignment can still be low, showing that there are examples where the parameter is present in the instruction but it does not match the parameter in the ground-truth response.

B ICE

B.1 Full Prompt

In Figure 14 we provide the full prompt for our proposed in-context evaluation method. The prompt is constructed as follows: a description of the task, documentation of the APIs selected, one in-context example and the test queries.

B.2 ICE Score Calculation

To calculate the ICE score, we follow these steps:

1. We input to the model the ICE prompt (Figure 14), containing an in-context example from the assessed dataset, and obtain the model output for each of the 7 test instructions.

2. For each test instruction, we calculate the Levenshtein distance between the generated

Instruction Examples	
From ToolBench	I'm planning a surprise birthday party for my best friend and I need some help. Can you find the email of a person named Emma Watson at google.com? Additionally, I want to find a formulated product by its registration number to use as a gift for my friend.
	My family and I are considering relocating to New York City. Can you provide us with a list of transactions for zipcode 10019? We would like to see the last sales date, last sales amount, and total records for each transaction. Additionally, could you give us the detailed historical transactions for the address 310 W 56th St, New York, NY 10019?
	I want to explore movies related to a specific genre. Can you discover movies in the genre with genreId '80' and provide me with the details of the first 10 results? Also, fetch the crew details for a random movie.
	I'm a basketball enthusiast and I want to know more about the players in the NBA. Can you fetch me the details of all the players? Additionally, provide me with a random Chuck Norris joke to lighten the mood.
	My friends and I are planning a trip to multiple cities and we need to estimate the cost of living. Can you provide us with a list of available currencies? Additionally, we would like to get a comprehensive list of cities, including their countries, to help us plan our itinerary.
From ToolAlpaca	I'm curious about quotes related to debugging. Can you find some for me? After that, please show me a list of all authors so I can learn more about their thoughts on programming.
	I want to add a catchy animation to my GitHub profile. Show me a list of font types available for use, and once I choose one, create a typing and deleting SVG with the text "I'm a software engineer" in 18-point font size, orange color, a typing speed of 80 ms, start delay of 500 ms, and a pause duration of 1 second.
	I'm thinking of going to Lansdowne Park this afternoon. Could you find nearby bus stops within a 300-meter radius with my current location at latitude 45.3967 and longitude -75.6858?
	My user profile still shows my old email address. Can you update it to my new one, "new_email@example.com"? Also, update my preferences to receive newsletters about datasets in the "economy" category.
	Can you personalize the email content for my subscribers based on their names? Use the template 'Holiday Greetings' and add subscriber data for Sarah, whose email is sarah@example.com and name is 'Sarah Smith'.

Table 12: Examples of instructions which our method found as **incoherent**, from the two examined datasets.

API-call sequence and the correct API-call sequence.

3. We average the Levenshtein distances calculated for all test instructions, resulting in a single score for each data instance.

Steps 1 to 3 are repeated for each of the data instances in the assessed dataset. Figure 2 shows the distribution of instance-level scores for the two assessed datasets.

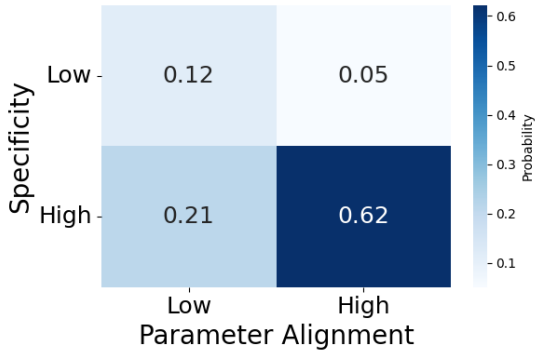
C Extrinsic Evaluation

C.1 Training Setup

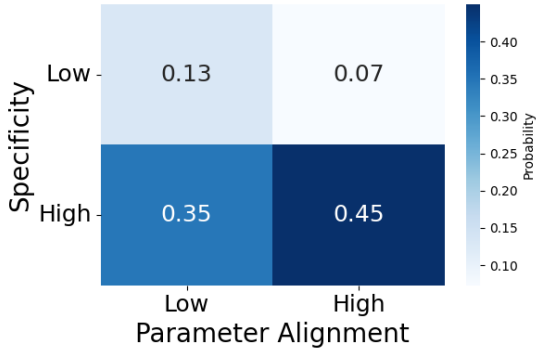
The training setup is similar for both ToolBench and ToolAlpaca benchmarks, where we train on pairs of (instruction, API-call sequence + response).

ToolBench. We fine-tune a LLaMA-7B model when working with the ToolBench dataset. The learning rate is set to 5×10^{-5}, and we use a batch size of 2. Since the tasks require relatively long inputs for the targeted model, the context length is extended using positional interpolation (Chen et al., 2023). We increase the context length to 4096, which is twice the model's default length of 2048. The model is trained for two epochs on 8 NVIDIA A10G Tensor Core GPUs.

ToolAlpaca. For the ToolAlpaca dataset, we fine-tune a Vicuna-7B model. We use a batch size of



(a) ToolAlpaca dataset



(b) ToolBench dataset

Figure 5: Confusion matrices for Specificity and Parameter Alignment.

2 and a learning rate of 2×10^{-5}. The model is fine-tuned for three epochs on 4 NVIDIA A10G Tensor Core GPUs.

C.2 Evaluation Setup

We adhere to the evaluation procedures outlined in the respective benchmarks for ToolBench and ToolAlpaca. Both benchmarks use a generative model for the evaluation of the API-call sequence and response. We use ChatGPT for both datasets.

ToolBench. In the ToolBench benchmark, the evaluation process begins with assessing the solvability of the given instruction. Using ChatGPT, solution paths are categorized as Pass, Fail, or Unsure based on this classification. The evaluation criteria include various rules to determine the success of a solution path. For more detailed insights into the evaluation methodology and rules, please refer to the original paper (Qin et al., 2024).

The original evaluation procedure involves assessing the generalization ability across three levels—unseen instructions, tools, and categories—as well as three different scenarios. However, instead of splitting the test set into categories, we calculate the pass rate by averaging over all test samples. Importantly, in the human-annotation of the test set, we aimed to maintain a similar distribution across all test splits for consistency.

Regarding the retrieval of APIs during model inference, we adopt only one of the approaches tested in the original evaluation, where we directly insert the relevant APIs for each test instruction. This approach simulates the scenario where the user specifies the preferred API set.

ToolAlpaca. Similarly, in the ToolAlpaca benchmark, we use ChatGPT to evaluate the model's output in addressing the instruction. The evaluation criteria is assessing the overall correctness, considered as the pass rate, of both the process and the response. For further details regarding the evaluation methodology, please refer to the original paper (Tang et al., 2023).

In our study, we use the simulated subset for evaluation. This subset comprises 10 simulated tools (100 instructions) that were not part of the training toolset. While the original paper also includes a real-world subset with 11 APIs from various domains, we focused solely on the simulated data due the lack of detailed instructions on how to use the real-world data.

Specificity – Extraction Task Prompt

```
You are data quality researcher in natural language. You
will be provided with user instruction to an AI
assistant. This instruction might include one or more
requests.

Extract the parameter values from the instruction. Use
the API documentation and choose only important
parameters required for the API, if there are any. Use
#missing for parameter values that are not explicitly
provided in the query.

Example 1:
Query: "Can you fetch the flight data for the company
AZU on June 15th, 2022?"
Required Parameters: [company, date]
Start Answer:
company = 'AZU'date = 'June 15th, 2022'

Example 2:
Query: "Can you fetch the flight data for the company ?"
Required Parameters: [company, date]
Start Answer:
company = #missing
date = #missing
```

Figure 6: Prompt for instruction **specificity**, as an extraction task.

Coherence – Next Sentence Coherence Prediction Task

```
You are tasked with assess the contextual coherence between
pairs of sentences.
Contextual coherence refers to the likelihood that the second
sentence follows the first sentence in a coherent and logical
manner.
For each pair of consecutive sentences, predict whether the
second sentence is likely to follow the first sentence in a
coherent sequence. Consider the flow of information and
whether the second sentence is contextually appropriate
following the first sentence.

Examples:

Query:
1. "I'm a cryptocurrency trader, and I want to analyze the
   historical prices and market caps of popular
   cryptocurrencies like Bitcoin, Ethereum, and Stellar."
2. "Can you fetch this information for me using the Crypto
Prices API?"
3. "Additionally, I'm planning a trip to North America and I
would like to know the subregions in North America using the
Geography API."
Answer:
1-2 = coherent
2-3 = incoherent

Query:
1. I want to impress my friends with some hilarious jokes.
2. Can you fetch some Chuck Norris jokes related to 'sports'?
3. Also, fetch me some Chuck Norris jokes related to 'art'?
Answer:
1-2 = coherent
2-3 = coherent
```

Figure 7: Prompt for instruction **coherence**, as a next sentence coherence prediction task.

Solvable Prompt

```
You will be given a user query, and a list of API functions
that can return external information. Each API function is
shown with its domain, name, description, parameters and
output fields.

Determine whether any subset of the API functions could
provide all the information required to answer the query. No
need for one API to answer all the requests in the query,
multiple APIs can be used to answer different requests in the
query. It is alright if a parameter value is not explicitly
provided in the query. Use the information provided in
Domain, Name, Description, and parameters.

Answer with 'Yes' or 'No' and provide an explanation for your
answer.

Example 1:
Query: "I'm traveling with my family. Can you tell us what's
the weather like in Lisbon for tomorrow? Also, prepare for us
an itinerary"
API Function 1 - Domain: Weather. Name: get_weather.
Description: get weather by city and date. Parameters: city,
date. Output: weather_description.
API Function 2 - Domain: Traveling. Name: prepare_itinerary.
Description: prepare_itinerary. Parameters: city, duration.
Output: places_list.

Explanation: The get_weather function provides the weather
given a city and date. We do not know the date of tomorrow,
but we allow non-explicit parameter values. The
prepare\_itinerary can handle the second request. Duration is
not mentioned, but we allow non-specfic parameters.

Answer: Yes
```

Figure 8: Prompt for instruction **solvability**.

Parameter Alignment – Extraction Task

```
You will be given a user query, and a list of API functions
that are suggested to address the user requests, only a
subset of them is relevant to answer the query.
Each API function is shown with its domain, name,
description, parameters and output fields.

Extract the parameter values which are explicitly mentioned
in the query.

In query_extracted: you have to extract or infer the
parameter values from the Query.
If a parameter {param1} value is not mentioned in the query
write (no_param1).

Example 1:
Query: "I'm traveling with my family. Can you tell us
what's the weather like in Lisbon for tomorrow? Also,
prepare for us an itinerary"
Parameters for extraction: [city, date, duration]
query_extracted = [city='Lisbon', date='tomorrow',
duration='no_duration']

Example 2:Query: "Create a video from random fashion
images"
Parameters for extraction: [query, date]
query_extracted = [query='fashion', date='no_date']
```

(a) Step 1: parameter value extraction

Parameter Alignment – Comparison

```
You will receive sets of parameter names and their
corresponding values extracted from queries
(query_extracted). Additionally, you will be provided with
a sequence of API calls (api_extracted) along with
parameter names and values.
Your task is to determine whether the parameter values
extracted from the queries align with the parameters in the
api_extracted.

Example 1:
query_extracted = [city='Lisbon', date='17 March',
duration='no_duration']
api_extracted: [city='Lisbon', date='17-3']
Explanation: duration value  duration='no_duration' means
it is missing in query_extracted. It is also missing in
api_extracted. 'Lisbon' in both. Date 17-3 value match but
different foramt, which is acceptable.
Answer: Yes

Example 2:query_extracted = [name='name']
api_extracted: [name='Suzan']
Explanation: 'name' and 'Suzan' are different in
query_extracted and api_extracted.
Therefore, not matching.
Answer: No

If query_extracted has parameter_name='no_parameter_name'
and in api_extracted it is provided with a variable (e.g.,
parameter_name='word'), answer with No.
```

(b) Step 2: comparison

Figure 9: Prompts for assessing **parameter alignment** in the API-call sequence, as a two-step procedure.

API Matching and Minimal Calls Prompt

```
You will be given a user query, and a list of API functions that are
suggested to address the user requests, only a subset of them is
relevant to answer the query. Each API function is shown with its
domain, name, description, parameters and output fields. Determine
whether the API call sequences functionality solves the user query,
and whether it solves it with minimal number of calls or it has
redundant calls that are not needed to solve the query. Use the
information provided in Domain, Name, Description, and parameters. If
Description not provided, consider the API as not solving.
Each query has multiple requests. The API call sequence should
address all different requests. Answer with two steps:
calls_solves: {Yes/No} whether the API call sequence would fully
solve the user query.
minimal_calls: {Yes/No} whether the API call sequence solves the user
query with minimal number of calls (Yes) or it has redundant calls
that are not needed to solve the query (No).
Example 1:
Query: "Can you fetch random fashion images. Then, create a video
from these images"
API call sequence: [get_random_image(query='fashion'), get_video()]
All APIs Documentation: API Function 1 - Domain: Content. Name:
get_random_image. Description: get list of images from the internet.
Parameters: query. Output: image.
API Function 2 - Domain: Content. Name: get_video. Description: get
trending videos. Parameters: query. Output: video.
Explanation: get_random_image fetches random images. get_video does
not handle the second request of creating video.
calls_solves: No. minimal_calls: No
Example 2:
Query: "Fetch me latest news about NBA"
API call sequence: [get_news(), fetch_news()]
API Function 2 - Domain: NBA. Name: get\news. Description: get latest
news. Parameters: None. Output: text.
API Function 3 - Domain: Finance. Name: fetch_news. Description: get
latest news of trading. Parameters: None. Output: text.
Explanation: fetch_news() is redundant, the query does not ask for
finance and trading news.
calls_solves: Yes. minimal_calls: No
```

Figure 10: Prompt for **sufficiency** and **minimality** of the API-call sequence.

Specificity - Direct Questioning Prompt

```
You are a helpful data quality researcher. You will be
given data examples which are generated by an AI model for
the goal of training an AI assistant to use API tools. Each
data example is an instruction which should resemble real-
world scenarios. Your task is to estimate each instruction
quality in terms of completeness and specificity.

Evaluate the specificity of each instruction, considering
whether it contains all necessary information without gaps
for the AI assistant to respond with a use of an external
function call.

Answer with (Specific/Unspecific)
Specific: The instruction is highly specific and complete,
with no significant missing information. The parameters
values are mentioned explicitly.
Unspecific: The instruction lacks essential details or
variables. The parameters values are not mentioned
explicitly.

Example 1:
Query: "Can you fetch the flight data for the company AZU
on June 15th, 2022?"
Required Parameters: [company, date]
Query explanation: the company name and date are mentioned
explicitly.
Answer: Specific

Example 2:
Query: "Can you fetch the flight data for the company ?"
Required Parameters: [company, date]
Query explanation: the company name and date are not
mentioned explicitly.
Answer: Unspecific
```

Figure 11: Prompt for **specificity**, as a direct questioning task.

Coherence - Direct Questioning Prompt

```
You are a helpful data quality researcher. You will be given
data examples which are generated by an AI model for the goal
of training an AI assistant to use API tools.
Each data example is a single or multi-instructions which
should resemble real-world scenarios. Your task is to
estimate each instruction quality in terms of coherence.

Evaluate the extent to which the different requests or
components of the instruction are logically connected and
relevant to each other. Use a scale from 1 to 5 for
assessment, with 1 being the lowest score and 5 the highest.
1 (Poor): The components or different requests of the
instruction are highly disjointed, lacking a logical
connection.
2 (Limited): The components or different requests of the
instruction are somewhat related but exhibit notable
separation.
3 (Average): The components or different requests of the
instruction have a moderate level of coherence but still
possess some degree of separation.
4 (Good): The components or different requests of the
instruction are reasonably connected and logically related.
5 (Excellent): The components or different requests of the
instruction are highly coherent, with a strong logical
connection.

Examples:
Query: I'm a cryptocurrency trader, and I want to analyze the
historical prices and market caps of popular cryptocurrencies
like Bitcoin, Ethereum, and Stellar. Can you fetch this
information for me using the Crypto Prices API? Additionally,
I'm planning a trip to North America and I would like to know
the subregions in North America using the Geography API.
Explanation: Fetching crypto information is not related to
fetching geography information request.
Score = 1 (Poor)
```

Figure 12: Prompt for **coherence**, as a direct questioning task.

Parameters Alignment - Ranking Prompt

```
We need your help in data quality assessment. The data is
generated for the goal of training an AI assistant to use API
tools.You will be given an instruction and API call with
parameters. Your task to assess the match quality of called
parameters in the provided API calls compared to the specific
variables mentioned in the queries, considering the
possibility that the api call has added (hallucinated)
parameters or the api call has missing called parameters.
Rate parameter matching as follows:
 3 (High): If the API call parameters fully align with the
specific needs mentioned in the query.
 2 (Medium): If there is a moderate alignment, low error of
added or missing parameters.
 1 (Low): If the API call parameters have a low or minimal
alignment with the specific needs mentioned in the query, few
added parameters not mentioned in the query, or missing
parameters mentioned in the query.

Examples:

Query: "Can you fetch the flight data for the company AZU on
June 15th, 2022?"
API-Call: get_airline_data(company='AZU', date='15-6-2022').
Explanation: company and date parameter values in the API
call match the variables mentioned in the query.
Score: 3 (High).

Query: "Can you fetch the flight data for the company AZU on
June 15th, 2022?"
API-Call: get_airline_data(company = 'AZU').
Explanation: API Call did not include the date in the called
parameters.
Score: 1 (Low).
```

Figure 13: Prompt for **parameter alignment**, as a direct questioning task.

ICE Prompt

You are AutoAPI, a virtual assistant specialized in generating API calls for various tasks.

Your goal is to assist users with their requests by selecting the correct API calls and filling in the appropriate parameters. You can use multiple api calls

Below is a list of potential tools, APIs, and parameters that you can use for generating API calls:

Potential Tools, APIs, and Parameters:

1. Travel Planner APIs

API: generate_itinerary. Parameters: destination (string), start_date (string), end_date (string), number_of_participants (integer, optional)

API: get_weather_forecast. Parameters: location (string), date (string, optional)

2. Language Translation API

API: translate_text. Parameters: original_text (string), target_language (string), source_language (string, optional)

3. Sports Statistics API

API: get_player_stats. Parameters: player_name (string), year (integer), team (string, optional)

4. Health Tracker APIs

API: track_caloric_intake. Parameters: food_items (array of string)

API: recommend_meal. Parameters: meal_type (string), vegetarian (boolean, optional)

5. Information Extraction.

API: extract_from_context. Parameters: text (string), query (string, optional)

6. Math Operations API

API: perform_math_operation. Parameters: operation (string), operands (array of numbers)

{in_context_example}

Test Queries:

Query 0: {query_example}

Query 1: I have this document path/summary.txt, could you tell me the final result?

Query 2: Create a personalized travel itinerary for me. I'm planning a trip to Barcelona, and I'll be there from August 15th to August 25th, 2023.

Query 3: I'm craving a tasty vegetarian meal for dinner. Any recommendations?

Query 4: How much of calories is a salad, grilled chicken, and a banana.

Query 5: What is 238*17? Also calculate 44 + 634 please.

Query 6: What's the weather like in Lisbon for tomorrow? What about Porto?

Query 7: Translate 'Hello, how are you?' to Spanish. And translate 'How old are you' to French.

Answer in the format:

Query i:

Query i API call: api_call().

Remember you can use multiple api calls.

Remember to choose the matching API from the list above with the correct parameters. use maximum 3 api calls. If you suggest more than 3 api calls, move to the next query! Don't write #4_start.

{task_examples}

Figure 14: The prompt used for in-context evaluation of a training instance (marked as `{in_context_example}` in the prompt).