FANTAstic SEquences and Where to Find Them: Faithful and Efficient API Call Generation through State-tracked Constrained Decoding and Reranking

Zhuoer Wang[†1] **Leonardo F. R. Ribeiro**[2]
Alexandros Papangelis[2] **Rohan Mukherjee**[2] **Tzu-Yen Wang**[2]
Xinyan Zhao[2] **Arijit Biswas**[2] **James Caverlee**[1] **Angeliki Metallinou**[2]
[1]Texas A&M University [2]Amazon
wang@tamu.edu

Abstract

API call generation is the cornerstone of large language models' tool-using ability that provides access to the larger world. However, existing supervised and in-context learning approaches suffer from high training costs, poor data efficiency, and generated API calls that can be unfaithful to the API documentation and the user's request. To address these limitations, we propose an output-side optimization approach called FANTASE. Two of the unique contributions of FANTASE are its State-Tracked Constrained Decoding (SCD) and Reranking components. SCD dynamically incorporates appropriate API constraints in the form of Token Search Trie for efficient and guaranteed generation faithfulness with respect to the API documentation. The Reranking component efficiently brings in the supervised signal by leveraging a lightweight model as the discriminator to rerank the beam-searched candidate generations of the large language model. We demonstrate the superior performance of FANTASE in API call generation accuracy, inference efficiency, and context efficiency with DSTC8 and API Bank datasets. Our code is publicly available at https://github.com/Edillower/FANTASE.

1 Introduction

In recent year, there has been a surge of interest in enabling the automated tool-using capability of intelligent systems (Schick et al., 2023; Mialon et al., 2023). Specifically, as a bridge to the larger world, Application Programming Interface (API) calls allow virtual assistants to control smart-home devices, retrieve information, make reservations, and more on the user's behalf. Figure 1 shows how an API call may improve the user-assistant conversation and satisfy the user's needs. Generating such an API call requires advanced capabilities in understanding the requirements of an API (including

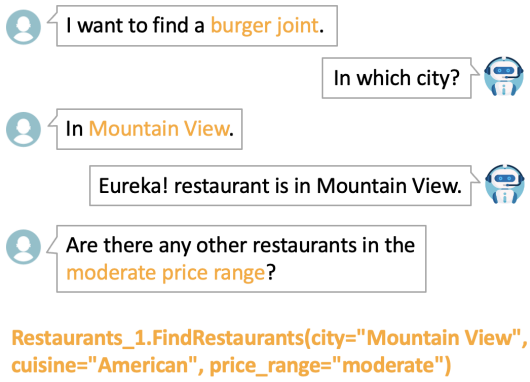

Figure 1: Example of an API call that retrieves information based on the user's needs given in the conversation.

its endpoints, parameters, and expected data formats) and reasoning over the conversation context to translate the user's needs into the appropriate API format.

With recent breakthroughs in generative Large Language Models (LLMs) such as GPT-X (Ouyang et al., 2022; OpenAI, 2023) and LLaMA (Touvron et al., 2023a,b), researchers have started to investigate their competence in complex reasoning tasks such as utilizing appropriate API tools (Li et al., 2023; Qin et al., 2023; Wang et al., 2023a). Their attempts focus on methods that can generally be grouped into those based on *supervised fine-tuning* for task-specific usage and those based on augmenting input-side context information (such as API shortlisting and exemplar selection) and optimizing prompts for *in-context learning* (Brown et al., 2020; Wei et al., 2022). Despite the strong supervision or extensive context, these methods still cannot ensure the generation's faithfulness with respect to the API documentation and suffer data and compute inefficiency. In contrast to previous works, we focus on how decoding strategies improve the generation's faithfulness, which is complementary to supervised fine-tuning and in-context learning methods. As a result, we present FANTASE (FANTAstic SEquences and Where to Find Them), a framework that employs State-tracked Constrained

[†]The major portion of the research was done during an internship at Amazon.

Findings of the Association for Computational Linguistics: EACL 2024, pages 6179–6191
November 12-16, 2024 ©2024 Association for Computational Linguistics

Decoding (SCD) and Reranking components, for faithful and efficient API call generation.

The SCD tracks the states of the generation and retrieves appropriate API documentation constraints in the form of Constrained Token Search Trie (CTST) used at each decoding step. SCD is guaranteed to generate API calls that are faithful with respect to the API documentation (§ 4.1), and provides inference efficiency (§ 6.2) with CTST that eliminates unnecessary forward inference passes. Compared to supervised fine-tuning methods, SCD brings considerable improvements (§ 6.1) without the data labeling and model training related hefty costs of labor, time, and computing that become increasingly expensive as the size of the LLM grows (Yang et al., 2023). SCD also reduces the in-context learning's reliance on the repeated supply of extensive contextual information for the inference of each instance (§ 6.3) by effectively incorporating API documentation constraints and guaranteeing the associated faithfulness at the decoding stage.

The Reranking component of FANTASE leverages models that are significantly smaller than LLMs for efficient incorporation of supervised signals (§ 4.2). As the correct API generation may not always have the highest sequence probability among beam-searched candidate sequences (§ 3), we train lightweight models to discriminate and rerank LLMs' candidate generations and demonstrate their effectiveness in digging out those correct sequences (§ 6.1). Compared to the supervised fine-tuning of LLMs, the Reranking component features extremely low training costs as it employs lightweight models. Compared to input-side optimized in-context learning methods, the Reranking component can address the severe performance issue associated with the absence of valuable supervised signals. Notably, FANTASE is a approach that suits the evolving and vast nature of real-world APIs. With the update of API documentation or the application to the new domain, LLMs fine-tuned with old data would require re-tuning with new data (Kumar et al., 2022). For FANTASE, SCD can adapt by constraining the decoding with a new set of constraints elicited from the new API documentation, while re-tuning the lightweight Reranking models has lower time and compute cost.

In summary, we make the following novel contributions:

- We propose State-tracked Constrained Decoding that can effectively enforce constraints elicited from API Documentation, which yields faithful generation and context efficiency.
- We leverage Constrained Token Search Trie to reduce unnecessary forward inference passes, which yields faster generation speed.
- We demonstrate the effectiveness of incorporating supervised signals with a small model to discriminate and rerank the beam-searched candidate generations of LLMs.

2 Related Work

Constrained Decoding offers controllable text generation by enforcing certain constraints at the decoding stage. Early research (Hokamp and Liu, 2017; Post and Vilar, 2018) concentrated on lexical constraints that enforce the inclusion of specific words or phrases in the outputs, which often neglects broader syntactic or semantic relationships. Later on, Lu et al. 2021 introduced *NeuroLogic Decoding* that handles more complex lexical constraints expressed by predicate logic. The subsequent extension, *NeuroLogic A*esque Decoding* (Lu et al., 2022), incorporated a lookahead heuristic to estimate future lexical constraint satisfaction. More recently, Chen et al. 2022 and Bastan et al. 2023 proposed parsing-based constrained decoding algorithms that tackle the challenge of ensuring correct syntactic relationships between word pairs.

Specific to structured text generation, Scholak et al. 2021 targeted Text-to-SQL generation and introduced *PICARD* that checks the validity at each decoding step for SQL lexical and grammar correctness with incremental parsing. The latest advancement was made by Geng et al. 2023 who demonstrated that an incremental parser can be used with formal grammar on a much wider range of structured NLP tasks without finetuning. While the results are encouraging, existing methods require post-hoc constraint satisfaction checking or rely on dependency parsing at inference time, or both, which compromises the efficiency. The most recent and closest work to ours is API-aware Constrained Decoding (Wang et al., 2023a) that imposes function and argument token constraints based on API documentation. However, despite limited improvements, its decoding strategy results in a 20% slowdown of the generation. In contrast to aforementioned methods, we achieve faster generation speed

Related Conversation	Human: I want to find a burger joint. Assistant: In which city?. Human: In Mountain View. Assistant: Eureka! restaurant is in Mountain View. ..... Human: Are there any other restaurants in the moderate price range?
Related API Documentation	..... Restaurants_1.FindRestaurants("cuisine" : Required, "city" : Required, "price_range" : Optional, "has_live_music" : Optional, "serves_alcohol" : Optional) ..... the possible values for "cuisine" include ["Mexican", "Chinese", "Indian", "American", "Italian"] .....
Expected API Call	`Restaurants_1.FindRestaurants(city="Mountain View", cuisine="American", price_range="moderate")`
Top Candidates	1. `Restaurants_1.FindRestaurants(price_range="moderate", city="MountainView")` missing `cuisine` 2. `Restaurants_1.FindRestaurants(cuisine="Burgers", city="MountainView")` missing `price_range` 3. `Restaurants_1.FindRestaurants(cuisine="American", city="MountainView")` missing `price_range` 4. `Restaurants_1.FindRestaurants(price_range="moderate")` missing `cuisine` and `city` 5. `Restaurants_1.FindRestaurants(cuisine="American", city="MountainView", price_range="moderate")`

Table 1: Preliminary analysis sample. With regular beam search decoding, the correct generation is only ranked the 5th, and other higher ranked generations exhibit various errors highlighted in red.

and guaranteed faithfulness with a novel State-tracked Constrained Decoding approach that dynamically incorporates appropriate constraints in the form of a retrieved token search trie.

Discriminator Guided Generation utilizes small discriminative models or external tools to guide the generation of LLMs. Dathathri et al. 2020 proposed the Plug and Play Language Model concept that guides the generation of pretrained models with a lightweight attribute classifiers' gradient. However, it increases compute costs due to the extra forward and backward passes required for sampling and using the gradients from the attribute classifiers to push the pretrained model's hidden activations. Following works including *GeDi* (Krause et al., 2021), *FUDGE* (Yang and Klein, 2021), and *BeamR* (Landsman et al., 2022) used different lightweight discriminators that classify the attribute of possible next tokens or partial sequence and reweigh token-level or beam-level probabilities at each decoding step towards the desired direction of attributes like sentiment, topic, formality, and so on. More recently, Ni et al. 2023 leveraged the execution results of a SQL executor to steer SQL generation, which achieved new state-of-the-art results. Nevertheless, the method is bounded by the prerequisite of the external executor. In our work, we employ a lightweight model to discriminate API call generation by the given context and perform a one-pass reranking of the beam-searched results, which brings in supervised signals effectively with little compute and time costs to the overall generation framework.

3 Preliminary Analysis

To better understand the capabilities and limitations of existing LLMs on the task of API call generation, we conduct a preliminary inference analysis on one hundred DSTC8 (Kim et al., 2019)[1] samples with an Alpaca (Taori et al., 2023) model that had been tuned with GPT-generated self-instruct (Wang et al., 2023b) data for better instruction following and in-context learning capabilities. We prompt the model with the DSTC8 data that contains task instruction, documentation of related APIs, two related exemplars, and conversation history. We use beam search with beam size 10 as the decoding algorithm, and we consider the top-10 high probability sequences as the candidate generations.

Our quantitative analysis shows that for 73% of the cases, the correct API calls are generated within those high probability sequences. However, within these cases, almost half of the correct sequences were not ranked as the highest, which yields a top-1 API call generation accuracy of 41%. Table 1 presents an example where the user wants to find a burger joint with a moderate price range in Mountain View. The supplied API documentation specified that the `Restaurants_1.FindRestaurants` function has cuisine and city as the required arguments, and the `cuisine` argument has five possible values. However, the correct sequence was only ranked the 5th for the given example. All the other 4 candidates that have higher sequence probabilities missed some required arguments, and the second one also wrongly generated `Burgers` instead of one of the five possible values for the argument `cuisine`. Note that the model demonstrates some reasoning capability that can correctly map `Burgers` into `American` as shown in the second and the fifth sequences. Nevertheless, the overall sequence probability favors the problematic generation of `Burgers`, which may be attributed to the explicit mention of the word in the given conversation history.

We conduct a further qualitative analysis to cate-

[1]Details will be given in Section 5.1

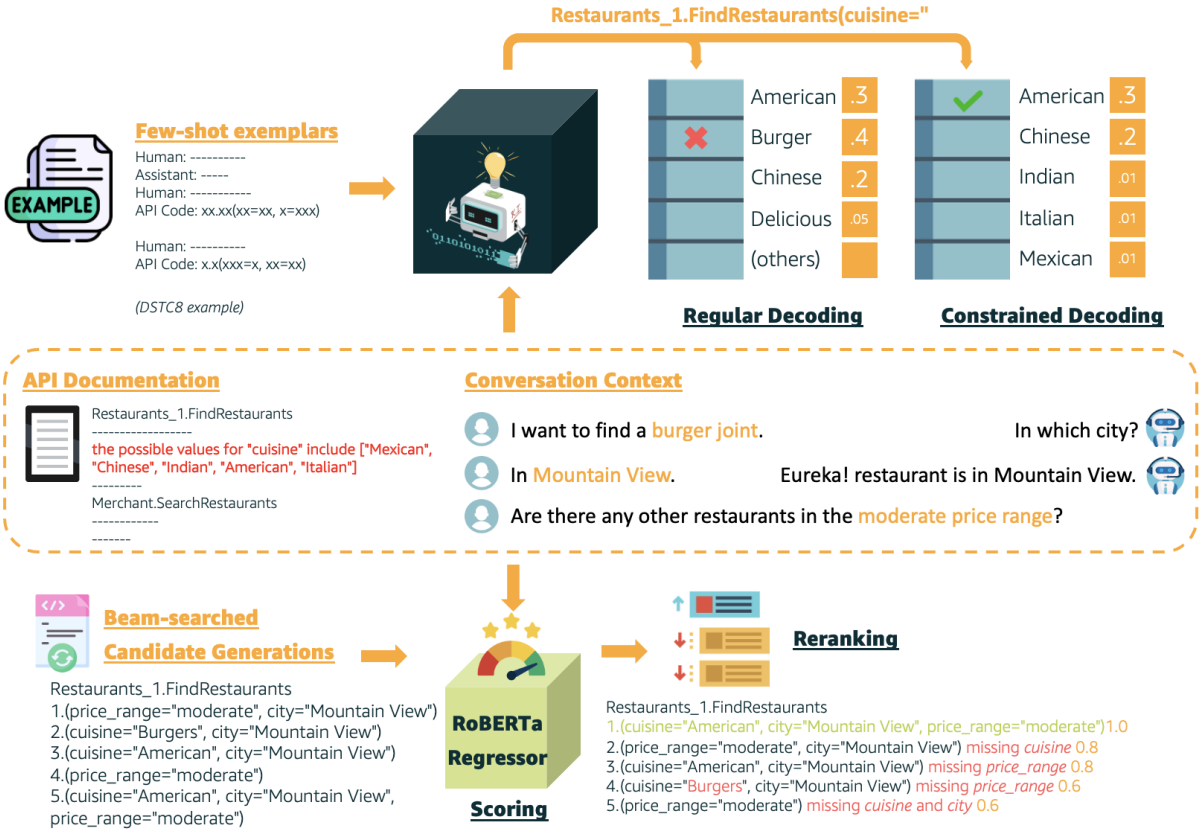

Figure 2: Illustration of the Concepts of Constrained Decoding and Reranking. (Upper half) Constrained Decoding enforces API documentation constraints and would only consider the five possible values of `cuisine`. (Lower half) A lightweight `RoBERTa` model is used to discriminate and rerank the beam searched candidate generations.

gorize the error types and possible mitigation for these cases. For the highest-ranked error cases, we find 33% argument value error, 24% missing required arguments, 19% missing optional arguments, 14% hallucination, and 10% argument name error. Furthermore, 42% of the error cases can be mitigated by enforcing the constraints described in the API documentation, 29% of the cases require the better understanding of the conversation, and the remaining 29% of the cases need a combination of the aforementioned two improvements.

4 The FANTASE Framework

In Section 3, we demonstrate that there are "FANTAstic SEquences" in the beam-searched candidate generations, and the question is where to find them. To dig out those "FANTAstic SEquences" with the data efficiency and compute efficiency in mind, we propose the FANTASE framework that consists of two major components – State-tracked Constrained Decoding (§4.1) and Reranking (§4.2), which aims at enforcing API constraints with guaranteed faithfulness to the API documentation and incorporating supervised signals at low compute costs respectively.

4.1 State-tracked Constrained Decoding

In Figure 2, we illustrate the concept of State-tracked Constrained Decoding (SCD). For a regular decoding step, the consideration of the entire vocabulary space would lead to the high probability of the word `Burgers` overshadowing the correct word of `American`. To ensure the faithfulness to the API documentation, our SCD approach enforces the model to only consider the probabilities of the five possible values as documented in the API documentation.

Different from conventional token-occurrence-based constrained decoding approaches as described in Section 2, our approach takes the relation between package, function, argument, and argument values into consideration. SCD allows precise and dynamic enforcement of constraints based on the API documentation and generated units, which avoids the look-ahead decoding and pruning as other constrained decoding algorithms would normally require. The implementation of SCD consists of three major parts: 1) extraction

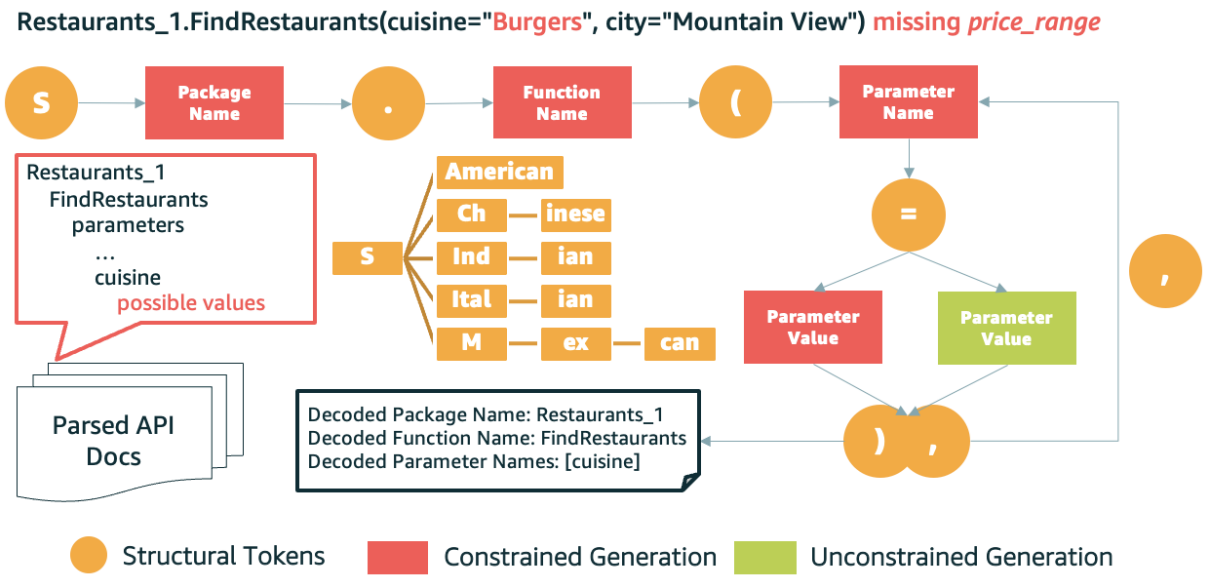

Figure 3: State-tracked Constrained Generation of API Call (showing the step of generating the value of parameter cuisine that has possible values of `American`, `Chinese`, `Indian`, `Italian`, and `Mexican`).

Structural Token	Generation State	Actions
S (pseudo)	Start of the generation	- Retrieve all package names - Constrained generation of package name
. DOT	End of package name Start of function name	- Record decoded package name - Retrieve possible function names by using package name - Constrained generation of function name
(LEFT_BRACKET	End of function name Start of argument name	- Record decoded function name - Retrieve possible argument names by using package and function names - Constrained generation of argument name
= EQUAL	End of argument name Start of argument value	- Record decoded argument name - Retrieve possible argument values by using package, function, and argument names - Check if the argument only takes certain possible values – If so, constrained generation of argument value – If not, perform regular unconstrained generation
, COMMA	End of argument value Start of argument name	- Reuse previously retrieved possible argument names - Constrained generation of argument name
) RIGHT_BRACKET	End of the generation	- Check if the list of decoded argument name contains all the required arguments – If so, conclude the generation – If not, replace RIGHT_BRACKET with COMMA and enforce continued generation

Table 2: State Tracking with Structural Tokens and Associated Actions.

of constraints from API documentation, 2) state tracking of the generation for constraints retrieval, and 3) constrained decoding with token search trie.

As API documentation is usually well-structured, it is feasible to extract constraints with simple rules. As a preprocessing step, we use regular expressions to extract five types of constraints including 1) available packages, 2) functions of each package, 3) required arguments of each function, 4) optional arguments of each function, and 5) possible values of each argument. We store these constraints in a lookup table with package name, function name, and argument name as the query keys. At the inference stage, we query the lookup table to fetch corresponding constraints by decoded package name, function name, and/or argument name. If the API is evolved with changing constraints such as new required/optional arguments, changing names, changing possible values, etc., new constraints can be enforced effortlessly at the inference stage by reparsing the updated API documentation, which is less expensive than re-tuning the model with updated labeled data.

In Figure 3, we illustrate the SCD at the inference stage. To ensure appropriate constraints can be retrieved and enforced at the precise inference step of the generation, the state of the generation is determined by tracking the model-generated structural tokens. The structured nature of the API call results in signature tokens that indicate the end or start of different units of the API call as we specified in Table 2. Specifically, for the constrained gen-

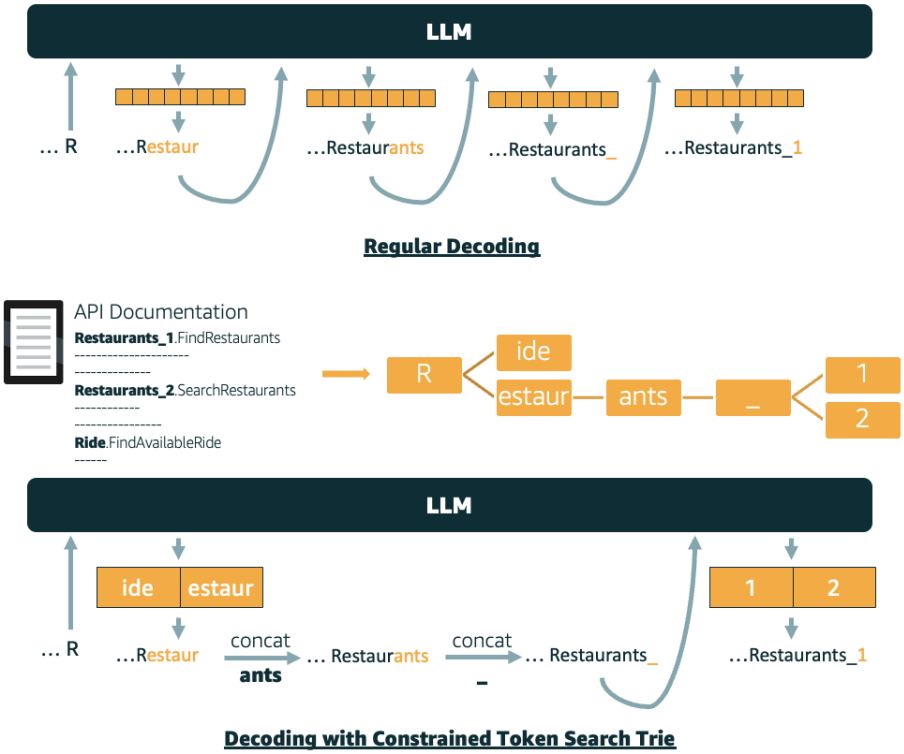

Figure 4: Comparison of the Generation of `Restaurants_1` with Regular Decoding and Decoding with Constrained Token Search Trie.

eration of an API call unit, we enforce the model to decode along the Constrained Token Search Trie (CTST) as illustrated in Figure 4. The tokenizer of LLMs performs WordPiece tokenization, which breaks down the word into smaller subword tokens for various benefits (Devlin et al., 2019). Accordingly, the package name `Restaurants_1` would be autoregressively generated by the LLM piece by piece with five forward-pass inference steps as shown in the upper half of Figure 4. At the preprocessing step, we build the extracted constraints into CTST. When conducting constrained generation, the forward inference pass is only necessary for nodes that have multiple branches. In such a case, only the probabilities of the possible next tokens as indicated by the CTST would be considered. For nodes that only have one child, the subsequent token is directly appended, which saves the time and compute costs of a forward inference pass.

For SCD, we implement it with four different sampling strategies including greedy search, top-k sampling, top-p sampling, and beam search.

4.2 Reranking

Although the SCD approach we introduced in Section 4.1 can guarantee the generated API call's faithfulness to the *API documentation*, the faithfulness to the *user's request* is solely dependent on the LLM's zero-shot or few-shot in-context learning and reasoning capabilities. Also, the valuable labeled training data hasn't been exploited yet with SCD. Fine-tuning the LLM might be a straightforward solution, but the huge compute costs associated with the growing parameter size of LLM motivates us to seek alternative solutions.

Inspired by the studies of discriminator-guided generation, we propose the supervised training of a lightweight scorer for the reranking of beam-searched candidate generations. To train the scorer, we generate data as follows: we prompt the Alpaca-13B model with training set samples and obtain associated beam-searched candidate generations for each sample. For each candidate generation, the matching score with respect to the ground truth is calculated as the target of the scorer. Such data is used for the tuning[2] of a RoBERTa-base(Liu et al., 2019) model that has 125M parameters to predict the matching score based on the input of conversation and API Documentation context and the candidate generation. Specifically, we train the model with sample-wise batching that groups candidate generations of the same context into one mini-batch and use the MSE Loss and Spearman Soft Rank-

[2]The rerankers were fine-tuned with 5 epochs, an effective batch size of 256, a learning rate of 5e-5, and AdamW as the optimizer.

ing Correlation Loss (Blondel et al., 2020) as the training objective for optimal performance.

Instead of returning the beam-searched candidate generation that has the highest sequence probability as the final generated API call, we use the trained scorer to discriminate each of the candidate generations and rerank them accordingly. This reranking strategy allows low-cost incorporation of task and domain-specific context reasoning capability learned from the valuable labeled data and complements the SCD approach and LLM's zero-shot or few-shot in-context learning and reasoning capabilities.

5 Experiment Setup

5.1 Datasets

We use the data from **DSTC8** (Kim et al., 2019) and **API Bank** (Li et al., 2023) to conduct the experiments and evaluation of our proposed FANTASE framework. Both datasets support the task of API call generation that requires understanding and reasoning of multi-turn human-assistant dialogues. Short-listed APIs and associated Documentation are accompanied by each sample. One major difference is that each DSTC8 sample has two related exemplars while API Bank does not ship with the exemplars. Accordingly, we experiment with the few-shot in-context learning setting with DSTC8 and the zero-shot in-context learning setting with API Bank. In Appendix A, we provide detailed statistics of DSTC8 and API Bank.

5.2 Baseline and Backbone Models

For in-context learning settings, we include strong baselines `GPT3.5-turbo` and `GPT4` developed by OpenAI, which represents the most recent breakthrough in LLMs with a track record of leading zero-shot and few-shot learning and reasoning capabilities. FANTASE is a plug-and-play model-agnostic approach that can be used in conjunction with any LLM that has an autoregressive decoder producing next-token probabilities. However, the API access of OpenAI models only allows greedy search and does not provide the logits. In consideration of the license, resource constraints, efficiency, and zero-shot/few-shot learning and reasoning capabilities, we opt to use `Alpaca-13B` as the backbone of our proposed methods. As for baselines of supervised learning settings, we fine-tune the `Alpaca-7B` model with DSTC8 training set samples (denoted as `AlpDSTC-7B`), and we directly use the `Lynx-7B` model, an API Bank data tuned `Alpaca-7B` model, released by Li et al. 2023. We supply detailed information of the aforementioned models in Appendix B.

5.3 Evaluation Settings

To verify the effectiveness of FANTASE, we run experiments with the following three evaluation settings that focus on different perspectives:

API Call Generation Accuracy measures if the generated API calls fully match their associated ground truth. It is the main metric that reflects if the generation faithfully followed the user's request and the requirements specified in the API documentation. As the order of arguments does *not* matter for both datasets, we calculate unit-wise order-insensitive set matches. The opposite order-sensitive case is, however, a setting bias in favor of FANTASE as the state-tracking of the SCD component makes it fully aware if it's going to generate the nth argument. For fair comparison, we follow Li et al. 2023 to report the order-insensitive matching results in this paper.

Inference Efficiency measures the time cost of the API call generation. Previous works on constrained decoding often vaguely report that the decoding speed is slower than regular decoding algorithms without quantitative measurements. To quantify the speed up brought by FANTASE's State-tracked Constrained Decoding that utilizes CTST (§4.1), we compare the time costs of the API call generation with regular/constrained greedy/beam search algorithms using the `Alpaca-13B` model under the in-context learning setting.

Context Efficiency measures the effectiveness of our approach in incorporating the API documentation without the reliance on the repeated supply of the lengthy API documentation in the prompt for in-context learning.

6 Results and Analysis

6.1 API Call Generation Accuracy

In Table 3, we report the results of API call generation accuracy. The SCD component of FANTASE consistently brings substantial improvements over the base models for both in-context learning settings and supervised learning settings on DSTC8 and API Bank, which demonstrates complementary benefits.

Specifically, for few-shot in-context learning settings evaluated with DSTC8, SCD Greedy Search

Dataset	DSTC8 (Two Exemplars)	API Bank (No Exemplars)
In-context Learning Methods		
Baselines		
GPT4	**51.33**	**63.66***
GPT3.5-turbo	49.28	59.40*
Alpaca-13B Greedy Search	37.63	24.06
Alpaca-13B Beam Search	40.49	24.31
FANTASE with Alpaca-13B as the Base Model		
SCD Greedy Search	42.33	56.64
SCD Beam Search	44.17	62.66
Supervised Learning Methods		
Baselines		
AlpDSTC-7B / Lynx-7B Greedy Search	46.63	48.62
AlpDSTC-7B / Lynx-7B Beam Search	47.44	50.53
FANTASE with Alpaca-13B as the Base Model (In-context) and RoBERTa-Base Reranker (Supervised)		
Reranking	46.42	33.33
SCD Beam Search + Reranking	48.88	64.41
FANTASE with AlpDSTC-7B / Lynx-7B as the Base Model		
SCD Greedy Search	59.30	65.66
SCD Beam Search	**62.78**	**67.17**

Table 3: API Call Generation Accuracy Evaluation On DSTC8 and API Bank. For settings involving beam search, we set the beam size to 4. For reproducible results, we set temperature to 0 for all settings. (* denotes results reported by Li et al. 2023. Best performed In-context and Supervised Learning Methods are **bolded**).

and SCD Beam Search improve the accuracy by +4.7 and +3.68 over the respective counterparts. For zero-shot in-context learning settings evaluated with API Bank, SCD Greedy Search and SCD Beam Search boost the accuracy by +32.33 and +34.34 respectively, which makes the 13B model's zero-shot generation performance comparable to the GPT3.5-turbo model that has an estimated parameter size of 175B and surpasses the accuracy of fine-tuned 7B model by a large margin. The larger performance gap signifies the value of SCD when labeled data is not available at all.

For supervised learning settings, SCD can still greatly improve the performance of corresponding settings of fine-tuned models and yields +17.04/+16.64 performance gain on API Bank and +12.67/+15.34 performance gain on DSTC8 with greedy/beam search, which leads to the 7B models outperforming GPT models that are much larger in terms of parameter size.

The Reranking component of FANTASE also brings considerable improvements over the base model by supervised training of a lightweight discriminator. The Reranking component alone improves the regular beam search results by +6.15 and +9.02 respectively on DSTC8 and API Bank. The FANTASE framework, with both the SCD component and Reranking component activated, achieves the best overall accuracy showing complementary benefits of the two components. Specifically, for DSTC8, the accuracy of 48.88 is close to the performance of GPT3.5-turbo and better than the supervised fine-tuned model AlpDSTC-7B. For API Bank, the accuracy of 64.41 is better than GPT4 and supervised fine-tuned model Lynx-7B.

6.2 Inference Efficiency

Decoding Strategy	DSTC8		API Bank	
	Inference Speed (sec/sample)	Speed Up	Inference Speed (sec/sample)	Speed Up
GS	5.32	-	5.85	-
SCD GS	3.42	x1.56	3.33	x1.76
BS	15.12	-	23.15	-
SCD BS	6.33	x2.39	10.27	x2.25

Table 4: Generation Speed of Regular Greedy Search (GS) and Beam Search (BS) Decoding versus FANTASE's State-tracked Constrained Decoding (SCD) Counterparts. DSTC8 is tested with A100 GPU, and API Bank is tested with A5000 GPU.

In Table 4, we quantitatively measure the inference time savings of the State-Tracked Constrained Decoding that leverages the CTST as we have introduced in Section 4.1 and illustrated in Figure 4. Compared to regular greedy and beam search, SCD has the capability of speeding up the API call generation by approximately 1.5x~2.4x times. When looking together with generation accuracy, it is quite encouraging that SCD greedy search can achieve regular beam search level's performance with significantly less amount of time and SCD Beam Search can achieve much better performance at regular greedy search level's time cost. For API Bank, SCD Greedy search can even outperform regular beam search with significantly less time cost.

6.3 Context Efficiency

Dataset	Setting	w. API Doc	w.o. API Doc	Δ
DSTC8	GS	37.63	33.74	-3.89
	SCD GS	42.33	40.70	-1.63
	BS	40.49	38.24	-2.25
	SCD BS	44.17	42.54	-1.63
API Bank	GS	24.06	4.76	-19.3
	SCD GS	56.64	22.81	-33.83
	BS	24.31	4.51	-19.8
	SCD BS	58.65	23.05	-35.6

Table 5: Generation Accuracy of Regular Greedy Search (GS) and Beam Search (BS) Decoding with / without API Documentation versus FANTASE's State-tracked Constrained Decoding (SCD) Counterparts.

As discussed in Section 4.1, another benefit of SCD is to save the context tokens required in the prompt for supplying the API documentation to the LLM, as SCD is capable of incorporating that information at the decoding stage. Based on the statistics provided in Appendix A, removing API documentation from the input could save an average of 766.28 tokens for DSTC8 and 265.71 tokens for API Bank. In Table 5, we show the performance of SCD when the API documentation is removed from the model's input. For DSTC8, our constrained decoding method manages to maintain the accuracy above `40` when the API Documentation is absent from the input while the unconstrained counterparts suffer larger performance drops of `-3.89`/`-2.25` that leads to larger performance gaps with our approach. For API Bank, the absence of both exemplars and API documentation makes it super challenging for the model to generate relevant API calls as shown by the large performance drop. However, our constrained decoding method can still achieve `22.81`/`23.05` accuracy in such a scenario, which is close to the performance of the unconstrained version that has API Documentation access in the prompt.

7 Conclusions

The FANTASE framework, with its State-Tracked Constrained Decoding (SCD) and Reranking components, effectively tackles the challenges of generating API calls from complex contexts. By integrating API constraints through a Token Search Trie and employing a lightweight model for reranking, FANTASE not only ensures accurate API call generation but also improves inference and context efficiencies. Its superior performance on the DSTC8 and API Bank datasets confirms FANTASE's significant advancement in enhancing large language models' tool-using ability.

Limitations

Despite the effort we have made to the best of our current ability, we recognize the following limitations of our work:

Evaluation with larger language models: Due to our resource limitations and the input length of the two datasets, the largest models we can fine-tune and infer are 7B and 13B respectively. The effect of the base model's parameter size hasn't been extensively studied in this paper, and the impact of our approach on larger models' API call generation accuracy and inference efficiency lacks empirical evidence. Based on the working mechanism of FANTASE, our educated guess is that larger models may make fewer errors which our approach is targeting, so the improvements would be smaller than using relatively small models as the backbone. As for the inference efficiency, the absolute time savings should be larger as a forward pass through a larger model takes longer time. However, the relative speed-up would remain at the current level as the amount of nodes that only have one child depends on the data instead of the models.

Adaptation to research-wise understudied API formats: As the API call generation datasets available in the research community are based on popular programming languages, especially Python, we implement and verify the effectiveness of FANTASE's SCD component following the conventional API format. Although the high-level idea of SCD is adaptable to research-wise understudied API formats, such adaptation may require moderate engineering effort to replace the structural tokens we used with the respective new format's structural tokens. However, given FANTASE's effectiveness and efficiency, we believe that the benefits would outweigh the adaptation costs especially when the intended use is for large-scale real-world customer-facing applications where accuracy and efficiency matter.

Adaptation to a broader range of tasks: FANTASE is specially designed and evaluated for the task of our interests - API call generation. Although the high-level idea and concepts of our approach should be adpatable to other structured text generation tasks such as SQL generation, table generation, etc., the adaptation may require considerable efforts in identifying constraints, signature tokens, and appropriate constraint-enforcing steps. For unstructured text generation tasks, it remains unclear if the identification of signature tokens and appro-

priate constraint-enforcing steps are feasible.

Ethics Statement

Our approach significantly enhances the capability of current models to generate API calls. However, it's important to acknowledge that the accuracy of these generations remains imperfect. As API calls could enable language models to perform tangible real-world actions, inaccuracies in API call generation would lead to serious consequences. These may include but are not limited to: financial losses when making wrong purchases and reservations, potential harm to the human being or the environment when controlling physical objects in unexpected ways, and the dissemination of false or misleading information when retrieving a wrong set of information. Consequently, we urge the users of our methods and the related API call generation models to be aware of such systems' high likelihood of generating inaccurate API calls and to remain vigilant about the possible risks associated with such errors.

References

Mohaddeseh Bastan, Mihai Surdeanu, and Niranjan Balasubramanian. 2023. NEUROSTRUCTURAL DECODING: Neural text generation with structural constraints. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9496–9510, Toronto, Canada. Association for Computational Linguistics.

Mathieu Blondel, Olivier Teboul, Quentin Berthet, and Josip Djolonga. 2020. Fast differentiable sorting and ranking. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Xiang Chen, Zhixian Yang, and Xiaojun Wan. 2022. Relation-constrained decoding for text generation. In *Advances in Neural Information Processing Systems*, volume 35, pages 26804–26819. Curran Associates, Inc.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2020. Plug and play language models: A simple approach to controlled text generation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. 2023. Grammar-constrained decoding for structured NLP tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10932–10952, Singapore. Association for Computational Linguistics.

Chris Hokamp and Qun Liu. 2017. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1535–1546, Vancouver, Canada. Association for Computational Linguistics.

Seokhwan Kim, Michel Galley, Chulaka Gunasekara, Sungjin Lee, Adam Atkinson, Baolin Peng, Hannes Schulz, Jianfeng Gao, Jinchao Li, Mahmoud Adada, Minlie Huang, Luis Lastras, Jonathan K. Kummerfeld, Walter S. Lasecki, Chiori Hori, Anoop Cherian, Tim K. Marks, Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, and Raghav Gupta. 2019. The eighth dialog system technology challenge.

Ben Krause, Akhilesh Deepak Gotmare, Bryan McCann, Nitish Shirish Keskar, Shafiq Joty, Richard Socher, and Nazneen Fatema Rajani. 2021. GeDi: Generative discriminator guided sequence generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4929–4952, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Ananya Kumar, Aditi Raghunathan, Robbie Matthew Jones, Tengyu Ma, and Percy Liang. 2022. Fine-tuning can distort pretrained features and underperform out-of-distribution. In *International Conference on Learning Representations*.

David Landsman, Jerry Zikun Chen, and Hussain Zaidi. 2022. BeamR: Beam reweighing with attribute discriminators for controllable text generation. In *Findings of the Association for Computational Linguis-*

tics: AACL-IJCNLP 2022, pages 422–437, Online only. Association for Computational Linguistics.

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

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Ximing Lu, Sean Welleck, Peter West, Liwei Jiang, Jungo Kasai, Daniel Khashabi, Ronan Le Bras, Lianhui Qin, Youngjae Yu, Rowan Zellers, Noah A. Smith, and Yejin Choi. 2022. NeuroLogic a*esque decoding: Constrained text generation with lookahead heuristics. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 780–799, Seattle, United States. Association for Computational Linguistics.

Ximing Lu, Peter West, Rowan Zellers, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. NeuroLogic decoding: (un)supervised neural text generation with predicate logic constraints. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4288–4299, Online. Association for Computational Linguistics.

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

Ansong Ni, Srini Iyer, Dragomir Radev, Ves Stoyanov, Wen-tau Yih, Sida I Wang, and Xi Victoria Lin. 2023. Lever: Learning to verify language-to-code generation with execution. In *Proceedings of the 40th International Conference on Machine Learning (ICML'23)*.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *NeurIPS*.

Matt Post and David Vilar. 2018. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1314–1324, New Orleans, Louisiana. Association for Computational Linguistics.

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. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *ArXiv*, abs/2302.04761.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

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. 2023a. Llama: Open and efficient foundation language models.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models.

Shufan Wang, Sébastien Jean, Sailik Sengupta, James Gung, Nikolaos Pappas, and Yi Zhang. 2023a. Measuring and mitigating constraint violations of in-context learning for utterance-to-API semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 7196–7207,

Singapore. Association for Computational Linguistics.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023b. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508, Toronto, Canada. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Bing Yin, and Xia Hu. 2023. Harnessing the power of llms in practice: A survey on chatgpt and beyond.

Kevin Yang and Dan Klein. 2021. FUDGE: Controlled text generation with future discriminators. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3511–3535, Online. Association for Computational Linguistics.

Appendix

A Datasets

Dataset	DSTC8 median / mean	API Bank median / mean
Total Input Length (Tokens)	1,683 1,644.28	492 542.86
API Documentation Length (Tokens)	735 766.28	244 265.71
Exemplars Length (Tokens)	580 558.00	N/A
Conversation Length (Tokens)	200.5 221.08	138 175.15
Conversation Turns	6 6.31	3 3.21
Target API Call Length (Tokens)	46 45.70	27 35.17
Target API Call Arguments Amount	3 3.42	2 2.27

Table 6: Statistics of DSTC8 and API Bank Data.

In Table 6, we provide the detailed statistics of DSTC8 and API Bank. The inputs and outputs of DSTC8 are longer than API Bank counterparts. DSTC8 also has more turns of conversation and API call arguments than API Bank, which makes DSTC8 a more challenging dataset in terms of reasoning the context and generating appropriate API calls even if two exemplars are provided for each sample.

Our experiments and evaluation of the FANTASE framework are based on the test split of DSTC8 and API Bank that has 490 and 399 samples respectively. For DSTC8, we remove the longest one out of the 490 samples as it causes CUDA out of memory error on our server.

B Models

`Alpaca-13B` is a `LLaMA`-based instruction following model that has 40 layers, a hidden size of 5120, 40 self-attention heads and 13 billion parameters. For our experiments, we use Huggingface checkpoint `chavinlo/gpt4-x-alpaca`.

`AlpDSTC-7B` and `Lynx-7B` are the `Alpaca-7B`-based model that has 32 layers, a hidden size of 4096, 32 self-attention heads and 7 billion parameters. `Lynx-7B` is the training set fine-tuned model released by API Bank authors (Huggingface checkpoint: `liminghao1630/Lynx-7b`). The model was tuned for 3 epochs with a learning rate of 2e-5 and an effective batch size of 256. We follow the same setting of `Lynx-7B` to tune a `Alpaca-7B` with DSTC8 training set samples.

`RoBERTa-base` is a `BERT`-based model that has 12 layers, a hidden size of 768, 12 self-attention heads, and 125 million parameters. For our experiments, we use Huggingface checkpoint `roberta-base` and tune the model with training set candiate generations for 5 epochs with a learning rate of 5e-5 and an effective batch size of 256.

`GPT3.5` and `GPT4` are only accessible via API or web interface. Details of these two models have not been officially released by OpenAI, but a broadly accepted latency-based parameter size estimation is 175 billion parameters for `GPT3.5` and 1.76 trillion parameters for `GPT4`. For our experiments, we use checkpoints `gpt-3.5-turbo-0613` and `gpt-4o-2024-05-13`.

License information of `Alpaca` can be found at https://github.com/tatsu-lab/stanford_alpaca/blob/main/LICENSE and https://github.com/tatsu-lab/stanford_alpaca/blob/main/DATA_LICENSE.

License information of `LLaMA` can be found at https://docs.google.com/forms/d/e/1FAIpQLSfqNECQnMkycAp2jP4Z9TFX0cGR4uf7b_fBxjY_OjhJILlKGA/viewform.

License information of `Lynx-7B` and API Bank data can be found at https://github.com/

Method	Labeled Data Amount	Learning Efficiency	Contex Requirement	Faithfulness to API Documentation	Generation Accuracy	Generation Efficiency
Fine-tuning with Regular Decoding	Large	Worst	Low	Not Guaranteed	Better	Worst
Fine-tuning with FANTASE-SCD	Large	Worst	Low	Guaranteed	Best	Best
In-context Learning with Regular Decoding**	Small or Zero	Best	High	Not Guaranteed	Worst	Worst
In-context Learning with FANTASE-SCD	Small or Zero	Best	Moderate	Guaranteed	Better	Best
In-context Learning with FANTASE-SCD and Fine-tuned Reranker	Large	Better	Moderate	Guaranteed	Much Better	Better

Table 7: Comparative Summarization (** Summarized from the results and analysis of the 13B model for fair comparison)

AlibabaResearch/DAMO-ConvAI/blob/main/api-bank/LICENSE.
License information of DSTC8 data can be found at https://github.com/google-research-datasets/dstc8-schema-guided-dialogue/blob/master/LICENSE.txt.
License information of RoBERTa can be found at https://github.com/facebookresearch/fairseq/blob/main/LICENSE.
License information of GPT-X models can be found at https://openai.com/policies/terms-of-use.

C Comparative Summarization

In Table 7, we summarize the pros and cons of different methods based on the experiments and analysis we have reported in this paper for a more nuanced understanding of FANTASE's strengths and weaknesses in relation to other approaches.

D Decoding Illustration

A step-by-step walkthrough of the complete inference process for the sample given in Figure 3 is available on pages 53-68 of our presentation slides (https://edillower.github.io/assets/pdf/FANTASE-slides.pdf).