Hierarchical and Dynamic Prompt Compression for Efficient Zero-shot API Usage

Yichen Jiang[*1] **Marco Del Vecchio**[2] **Mohit Bansal**[1] **Anders Johannsen**[2]
[1]UNC Chapel Hill [2]Apple
{yichenj}@cs.unc.edu

Abstract

Long prompts present a significant challenge for practical LLM-based systems that need to operate with low latency and limited resources. We investigate prompt compression for zero-shot dialogue systems that learn to use unseen APIs directly in-context from their documentation, which may take up hundreds of prompt tokens per API. We start from a recently introduced approach (Mu et al., 2023) that learns to compress the prompt into a few "gist token" activations during finetuning. However, this simple idea is ineffective in compressing API documentation, resulting in low accuracy compared to the baseline using an uncompressed prompt. In this work, we introduce two major improvements. First, we specialize gist tokens for different hierarchies within an API: we use one Gist_{arg} token for compressing an argument and one $\text{Gist}_{\text{value}}$ token for compressing an acceptable value of a categorical argument. We then dynamically reveal $\text{Gist}_{\text{value}}$ tokens only when they are needed. Second, we add a reconstruction loss to predict the API documentation from the gist tokens. On multiple API-calling tasks, our proposed system keeps the simplicity, efficiency, and large compression factor (20x on SGD) of the gist token approach while achieving significantly better accuracy.[1]

1 Introduction

Large Language Models (LLM) have been shown to be able to use external tools or APIs in a zero-shot manner by in-context learning from APIs' documentation (Shen et al., 2023). Specifically, the LLM is given a prompt that includes a detailed description of an API's functionality and its acceptable arguments and values. It is also presented with a user's request or a conversation between the user and the system. The model is then asked to generate an API call that covers all the user's requests so far. We show an example in Fig. 1.

Despite the benefits of learning new APIs in-context, deploying such a model is challenging for latency-critical and resource-constrained settings. This is partially because of the time and memory it takes to compute the attention weights between the newly generated token and all tokens in the API documentation (Pope et al., 2023). For example, generating an API from the documentation of 103 tokens using LLaMA (Touvron et al., 2023) 7B costs an extra of 42 ms, 1729 GFLOPS of compute and 9.1 GB memory compared to generating it without the documentation.[2] In this work, we aim to accelerate the generation of the API call by compressing the documentation into **H**ierarchical and **D**ynamic "HD-Gist tokens". *First, we propose a scheme to compress an API documentation* ***hierarchically***: we insert one "argument gist token" (Gist_{arg} in Fig. 2b) after every argument's description; for those categorical arguments, we additionally insert one "value gist token" ($\text{Gist}_{\text{value}}$ in Fig. 2b) after every acceptable value of the argument. Intuitively, each argument is coarsely encoded into a Gist_{arg} token, while a categorical argument's values are additionally encoded into a set of $\text{Gist}_{\text{value}}$ tokens. We can train the proposed hierarchical HD-Gist model with no additional cost over the standard finetuning (following Mu et al. (2023)), by simply modifying the attention mask. The model first encodes the API documentation with the inserted gist tokens from left to right normally. Then, as the model encodes the user's conversation and generates the API call, we mask out all but those Gist_{arg} tokens. This encourages the model to compress the API documentation into gist tokens, that can then be attended to during the generation of the API call.

Second, we allow the model to 'zoom' in/out of a

[*]Work partially done while at Apple.
[1]Our code is publicly available at https://github.com/jiangycTarheel/HD-Gist.
[2]Benchmarked on a single NVIDIA A100 40GB.

Findings of the Association for Computational Linguistics: EACL 2024, pages 2162–2174
March 17-22, 2024

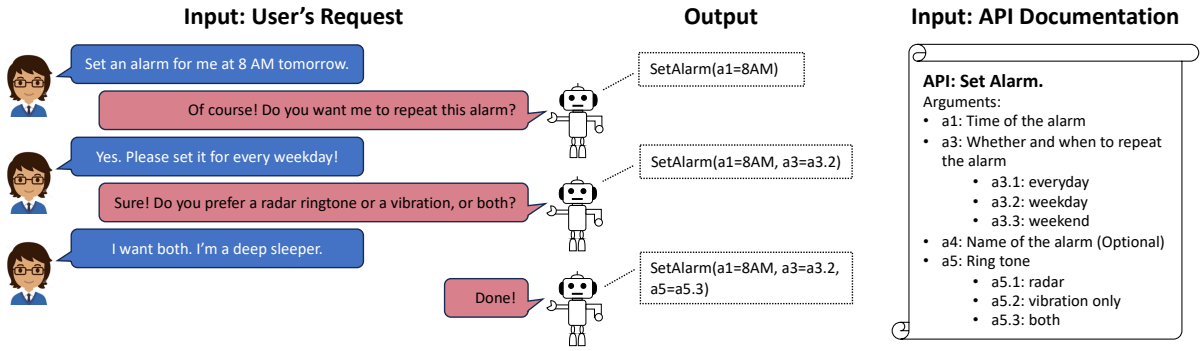


Figure 1: An example of the task discussed in this work. The model is given its conversation with the user and the API documentation. It then predicts the API call to fulfill the user's request.

compressed, categorical argument by ***dynamically*** *adjusting the gist mask*: the model unmasks the $\text{Gist}_{\text{value}}$ tokens of an argument right after it has generated this argument in the API call, and re-masks these tokens after it has predicted the value. *Finally, we also optimize the model to* ***reconstruct*** *the API documentation from the HD-Gist tokens only.* We again only unmask the $\text{Gist}_{\text{value}}$ tokens of an argument when the model is reconstructing its description and acceptable values. This reconstruction objective regularizes the model to hierarchically compress all crucial information about the arguments and values into the HD-Gist tokens.

We finetune a LLaMA 7B model (Touvron et al., 2023) with different compression methods to generate the API call in the SGD training set (Rastogi et al., 2020). We then evaluate the model on unseen APIs and conversations on the SGD and SGD-X (Lee et al., 2022) test sets. First, the proposed model with HD-Gist tokens obtains higher accuracy (56.68% on SGD and 54.83% on SGD-X) than any models with a fixed number of static gist tokens (41.43% on SGD and 39.53% on SGD-X with 20 gist tokens). Next, our experiments show that our reconstruction objective improves the accuracy of both the static gist model and HD-Gist model, while the HD-Gist model still maintains a sizable advantage (71.22% VS 46.47% on SGD). Notably, the proposed HD-Gist model is only 1.44% lower than the LLaMA baseline with uncompressed documentation. On the SGD-X test set, it even outperforms the LLaMA baseline by 4.5% in the accuracy, suggesting that compressing the documentation can even act as a regularizer to improve the out-of-domain generalization. We further show that HD-Gist is generalizable: on the APIBench dataset (Patil et al., 2023), it again achieves stronger results than all static gist models. Last but not least, the proposed method maintains a similar amount of compute and memory usage to the static gist model (Mu et al., 2023). Compared to the LLaMA baseline with uncompressed API documentation, using HD-Gist tokens achieves a 5.6% speedup in CUDA time, 29.9% reduction in compute, and 32.5% reduction in memory usage.

To understand the improvement of the proposed model, we also perform an error analysis on the SGD validation set. First and foremost, we find that static gist baseline predicts a wrong value for a categorical argument in more than 46% of the examples. Using the HD-Gist tokens can significantly reduce this error to 17% and adding the reconstruction loss further reduce it to 14%. Moreover, our proposed model also makes fewer errors in missing arguments, generating extra arguments, and hallucinating arguments that is not in the documentation.

Overall, by only attending to an average of 5.08 tokens in the API documentation per generation step, our proposed HD-Gist model significantly improves upon the previous state-of-the-art compression method. It closes the accuracy gap to the baseline that needs to attend to an average of 108.94 tokens in the API documentation, while requiring 30% less compute and memory usage.

2 Background and Related Work

Language models using external tools. With the recent tide of advancement in large language models (LLMs) comes further investigations into their weaknesses (e.g., incapability in math (Cobbe et al., 2021), hallucinating contents (Dziri et al., 2022), etc). Researchers have been trying to augment LLMs with external tools including web browsing (Nakano et al., 2021; Lazaridou et al., 2022; Komeili et al., 2022), calculators (Cobbe et al., 2021; He-Yueya et al., 2023), translation, code interpreters (Gao et al., 2023), or a combination of them (Thoppilan et al., 2022; Schick et al., 2023).

These preliminary efforts mostly focus on training/prompting LLMs to use a single tool or a limited pool of tools and cannot generalize to unseen tools without retraining or prompt engineering.

Zero-shot API usage by in-context learning from API documentation. More recent works (Shen et al., 2023; Liang et al., 2023) try to enable LLMs to use an infinite set of tools by exploring their ability to learn API documentation in-context and make API calls. Patil et al. (2023) introduced the APIBench dataset consisting of APIs from HuggingFace, TorchHub, and TensorHub and user requests. In this work, we also use the Schema-guided Dialogue (SGD) dataset (Rastogi et al., 2020) that challenges models to track dialogue states from a user-system dialogue following a schema of the service required. Based on the SGD dataset, Lee et al. (2022) further introduced SGD-X by rephrasing the API/argument's name and description. We use the schema of a specific intent (e.g., FindHotel) as the API documentation and ask the model to predict the value of all active arguments (e.g., check-in date) of the API.

Compressing prompts into gist tokens. In-context learning from API documentation enables LLMs to use potentially any tools. However, the length of API documentation grows with its complexity, including the number of acceptable arguments, different use cases, and so on. There has been a series of works that aim to compress Transformer's information-redundant, hidden activations into a small set of soft, compact vectors that can be used as the attention's keys and values in processing later tokens. Rae et al. (2020) first tried to compress activations using compression functions like pooling and convolution. Later works instead rely on the Transformer itself to compress a long sequence of activations into a shorter sequence of activations. Mu et al. (2023) proposed to append a few special "gist" tokens after the prompt and compress the prompt into the gist tokens' activations. The model can only attend to the gist tokens when encoding and decoding later context. This significantly speeds up the decoding, but at the cost of accuracy for knowledge-intensive tasks like API calling. More concurrent works (Jiang et al., 2023; Zhang et al., 2024) further improved upon gist-tokens in multiple aspects. For example, Ren et al. (2023) proposed to use a pair of sentinel tokens (similar to gist tokens) to mark the boundary of the span to be compressed, and achieve a wide range of compression ratios in a longer context. Chevalier et al. (2023) finetuned LLMs to compress segments of long context into individual memory vectors. To reduce information loss in compression, Ge et al. (2023) compressed long context into a few "memory tokens" using an additional LLM encoder. They pretrained this encoder with a fixed LLM decoder on language modeling and reconstruction objectives. They then finetuned the encoder on instruction-following data. In an alternative direction, Li et al. (2023) proposed "Selective Context" to directly prune redundant content in a given input context. Jung and Kim (2023) compressed the prompts with reinforcement learning.

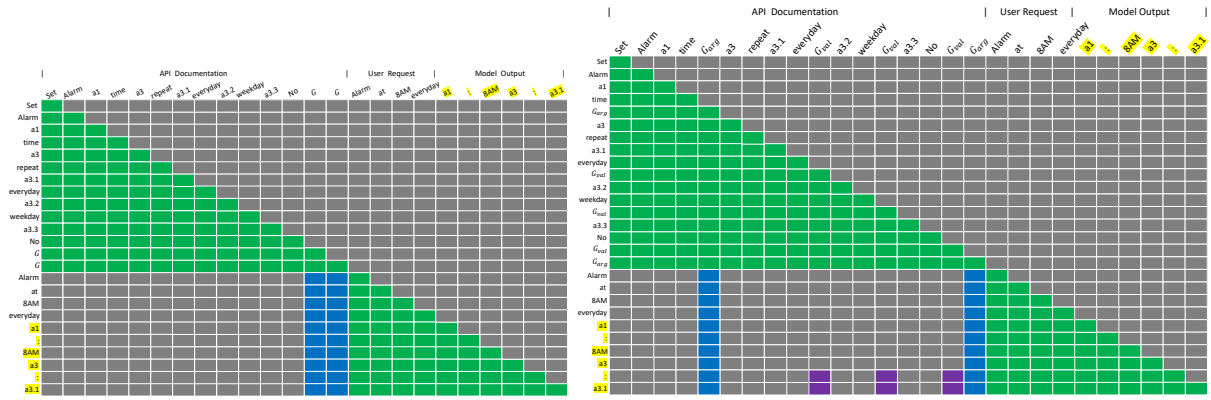
In a parallel direction, Xiao et al. (2023) proposed to keep a sliding window plus the 4 initial tokens' Key-Values in the cache as an "attention sink" during the inference. This method specializes in local language modeling at the cost of losing direct attention to distant contexts. In comparison, the gist-tokens methods focus on providing efficient but fine-grained access to distant context, which is essential in API-calling that requires copying specific parameter names. In this work, we inherit the lightweight compression method from Mu et al. (2023) that simply modifies the attention masks of tokens after the documentation without introducing a separate encoder. We further incorporate the auto-encoding objective (Ge et al., 2023) to improve the quality of compressed gist representations. Different from recent works that compress activations into static "gist" vectors, we introduce multiple sets of hierarchical and dynamic gist tokens to encode information at different granularities, and further allow automatic switching on/off a set to zoom in/out.

3 Method

In this section, we first explain the data preprocessing steps (Sec. 3.1). We then introduce the details about the HD-Gist tokens (Sec. 3.2) and reconstruct the API Documentation from HD-Gist (Sec. 3.3).

3.1 Preprocessing

Indexing the arguments and categorical values. When training a language model to follow API documentation, the model could quickly memorize the APIs in parameters and then operate independently of the documentation.This overfitting to seen APIs significantly harms the model's generalization to call unseen APIs at test time. To overcome this



(a) Static gist baseline (Mu et al., 2023).

(b) The proposed HD-Gist method.

Figure 2: The modified attention mask from the model with static gist (G) tokens (Mu et al., 2023) and our proposed model with HD-Gist tokens (including **hierarchical** Gist_{arg} and $\text{Gist}_{\text{value}}$). The causal attention mask shown is for a decoder-only model (e.g., LLaMA). Gray cells are zeros in the mask and other colored cells are ones in the mask. Blue cells represent attention to static gist tokens and argument-level Gist_{arg} tokens. Purple cells represent attention to the **dynamic**, value-level $\text{Gist}_{\text{value}}$ tokens. Model outputs are highlighted in curly brackets and yellow. We show the modified attention mask from HD-Gist model trained with the **reconstruction** loss in Fig. 3.

problem, we convert argument names and categorical values into structured indexes (e.g., "`a1`: Destination, `a3`: The number of stops in the itinerary, values=[`a3.1`: 1, `a3.2`: 0]"). The model is then asked to predict argument indices paired with either textual values or indexed categorical values ("`a1`='NYC', `a3`=`a3.2`"). This indexing scheme is based on the fact that argument and value names are simply *symbols* that can be replaced with anything, and it is the descriptions that actually encodes their meanings (Zhao et al., 2022).

Randomizing argument and value orders. Within an API documentation, we randomize the order of the arguments as well as the acceptable values of categorical arguments across different examples that share this API. This further prevents the model from memorizing the API documentation and helps the model to generalize.

3.2 Hierarchical and Dynamic Gist Token

Motivation. Recently, Mu et al. (2023) proposed to compress instructions into the activations of a few "gist tokens" inserted between the instruction ("Translate this into Spanish") and the input ("I like to play tennis."), by masking out the entire instruction after encoding the gist tokens. This method is lightweight as it only added an embedding vector of the gist token to the model parameters. However, we argue that appending all gist tokens sequentially after the API documentation may result in a loss of the hierarchical information. For example, for the `SetAlarm` API, the list of acceptable values "everyday; weekday; No" is relevant to the argument "repeat" only and is irrelevant to other arguments. Such hierarchy is originally encoded by the attention to the API documentation, but may get lost after being compressed into the gist tokens. We will later support this argument with a detailed error analysis (Sec. 5.3).

In order to retain this important hierarchy of API documentations during the compression, we introduce two major improvements to the static, sequential gist token method. *First, we propose a scheme to compress an API documentation* ***hierarchically***: we insert one "argument gist token" (Gist_{arg} in Fig. 2b) after every argument's description; for those categorical arguments (e.g., the "repeat" argument in the `SetAlarm` API), we additionally insert one "value gist token" ($\text{Gist}_{\text{value}}$ in Fig. 2b) after every acceptable value of the argument. Intuitively, each argument is coarsely encoded into a Gist_{arg} token, while a categorical argument's values is additionally encoded into a set of $\text{Gist}_{\text{value}}$ tokens. Following Mu et al. (2023), we can train the proposed hierarchical HD-Gist model with no additional cost over the standard finetuning, by simply modifying the attention mask. The model encodes the API documentation with the inserted gist tokens from left to right normally. Therefore, each gist token can possibly encode all preceding arguments. However, when the model encodes the user's conversation and generates the API call, we mask out all but those Gist_{arg} tokens. This encourages the model to compress the API documentation

into gist tokens, that can then be attended to during the generation of the API call.

*Second, we allow the model to 'zoom' in/out of a compressed, categorical argument by **dynamically** adjusting the gist mask.* When the model starts to generate the value for a categorical argument (e.g., it has generated "`a3:` "), it unmasks the $\text{Gist}_{\text{value}}$ tokens after every acceptable value (e.g., purple cells in Fig. 2). It then remasks these $\text{Gist}_{\text{value}}$ tokens after predicting the value (e.g., it has generated "`a3: a3.1`"). This in-context retrieval of $\text{Gist}_{\text{value}}$ tokens has two benefits: (1) it encourages the model to encode the fine-grained information about one categorical argument, exclusive of other arguments, into its $\text{Gist}_{\text{value}}$ tokens; (2) it avoids feeding the model with redundant tokens that would unnecessarily occupy memory and computation.

In summary, we insert HD-Gist tokens after different structures of the API hierarchy, and allow the model to dynamically switch on a set of $\text{Gist}_{\text{value}}$ tokens to zoom into a categorical argument.

3.3 Improving Compression Coverage by Learning to Reconstruct API

The existing objective supervises the model to generate the correct API call given the conversation with the user and the API documentation. However, in most examples, the API call only invokes some, but not all of the arguments in the documentation. Therefore, the existing objective does not provide the incentive for the model to compress all arguments in the gist token representations. To improve the completeness of the compressed API documentation, we add a second objective that trains the model to reconstruct the original API documentation given the argument-gist and dynamic value-gist tokens. Specifically, in all training examples, the model is given the API documentation and the conversation with the user and predicts the API call. In some training examples, we append a copy of the API documentation after the ground truth API call and a separator ([SEP]) token. When predicting the API documentation, the model can only attend to the argument and value gist tokens, while the model can additionally attend to the conversation when predicting the API call. We show the modified attention mask for an example with the reconstruction objective in Fig. 3.

4 Experiments

4.1 Experimental Setup

We adopt a unified setting across all datasets used in this work, in prompting an LLM to make API calls. The model's input consists of the API documentation and then the user request in the form of a single sentence or a conversation between the user and the system. Unlike the previous work (Patil et al., 2023), we put documentation before the user request because we need a static documentation representation that is independent of the user request. The model needs to generate a list of argument-value pairs that include all API arguments that the user has given a value. We conduct experiments in an oracle setting where the ground-truth API's documentation[3] is always given to the model in both training and evaluation without delegating to a API retrieval system as the impact of retrieval in a setting where the model is shown the k-best APIs is outside the scope of this work.

4.2 Datasets

SGD (Schema-Guided Dialogue) (Rastogi et al., 2020) is a public dataset in English that challenges models to perform dialogue state tracking (DST) by following a schema. We convert the original DST task into an API prediction task by (1) discarding arguments not used by the current API from the output and (2) giving the model the documentation of an API instead of a whole service. We train the model to predict an API call using the API documentation and the conversation between the user and the system. In both training and test, we also include intermediate turns where the user has not yet provided all arguments' values. In these turns, we ask the model to generate a partial API call with only those arguments mentioned by the user so far.

SGD-X (Lee et al., 2022) is created from SGD by asking human annotators to paraphrase the original arguments' names and descriptions into semantically similar yet stylistically diverse variants. SGD-X further evaluates models' robustness to linguistic variations in API documentation. We use the SGD-X/v5, which is the version with the most variation, as the extra test set to evaluate models trained on the original SGD training set.

[3] "Ground-truth API" refers to the API that can fulfill the user's request.

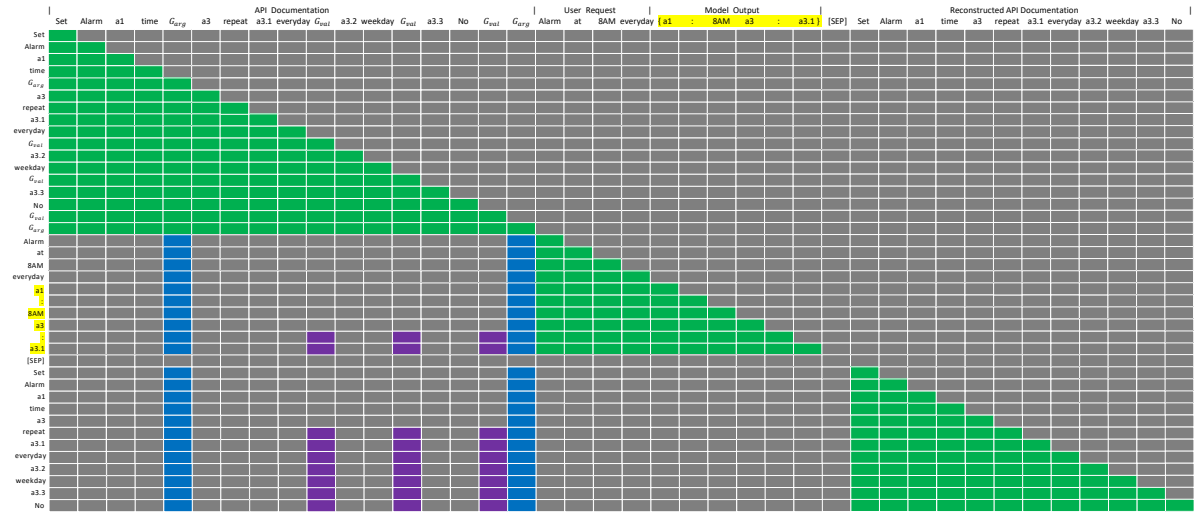


Figure 3: The modified attention mask from the model with the HD-Gist gist tokens that also reconstructs API documentation. There is a special token [SEP] that separates the API call and the reconstruction. After [SEP], the model can only attend to the Gist_{arg} tokens (blue cells) and preceding reconstruction (green cells). After the model has reconstructed the name of a categorical argument (`a3`), we unmask its $\text{Gist}_{\text{value}}$ tokens (purple cells) so that the model can access the encoded fine-grained information when generating its details, including all acceptable values.

APIBench (Patil et al., 2023) is a public dataset consisting of APIs from HuggingFace, TorchHub, and TensorHub as well as user question prompts in English generated from Self-Instruct (Wang et al., 2023). Because the sole purpose of this work is to train and evaluate models to follow API documentation, we discard the undocumented arguments of the API calls and only ask the model to predict (1) one out of three available APIs (HuggingFace, TorchHub, TensorHub), and (2) the value (pretrained model card's url) of the only argument of an API. Therefore, we insert an API-level gist token after every API, insert a $\text{Gist}_{\text{value}}$ token after the description of every model card, and omit the Gist_{arg} token since there is only one argument per API. We provide more details regarding the SGD and APIBench datasets in Appendix B.1.

4.3 Evaluation Metrics

Following Rastogi et al. (2020), we evaluate models on SGD and SGD-X using joint-goal accuracy. For arguments that are both in the ground truth and the predicted API call, we calculate the exact-match scores for values of categorical arguments, and calculate fuzzy soft-matching scores[4] for other arguments. A matching score of 0 is assigned for both of the following errors: (i) arguments that in the ground truth but missed in the prediction, (ii) arguments in the prediction but not the ground truth. The joint-goal accuracy is the product of matching scores of all arguments in the API documentation. For APIBench that has no non-categorical argument, we use the exact-match accuracy only.

[4]For example, predicting "New York" while the ground-truth is "New York City" results in a fuzzy-matching score of 0.76.

5 Results

5.1 Results on the SGD Datasets

Based on the results shown in Table 1, we can observe that the model with the argument-level gist tokens (Gist_{arg}) outperforms all static gist models (Mu et al., 2023) with up to 40 gist tokens (41.43% on SGD and 39.53% on SGD-X with 20 gist tokens). The proposed model with HD-Gist tokens obtains even better accuracy (56.68% on SGD and 54.83% on SGD-X) than all other models with compressed API documentation. This model only needs to store an average of 10.84 Gist_{arg} and $\text{Gist}_{\text{value}}$ tokens in memory, and attend to an average of 5.08 gist tokens per generation step. This is a significant reduction from the more than 108 tokens that need to be kept in memory and attended by the LLaMA baseline. We further supervise the models to reconstruct the API documentation from the gist tokens in 30% of the training examples. We observe that both the static and HD-Gist models benefit from this extra objective, while the proposed HD-Gist model still maintains a significant advantage (71.22% VS 46.47% on SGD and 69.24% VS 41.52% on SGD-X).

Models	API Doc Tokens in Attn	API Doc Tokens in Memory	Accuracy SGD	Accuracy SGD-X
LLaMA	108.94	108.94	72.66±1.7	64.78±0.7
Without Reconstruction Objective				
2 Gist	2	2	35.71±1.1	31.56±0.5
5 Gist	5	5	35.68±0.8	32.51±1.6
10 Gist	10	10	39.96±0.4	35.84±2.4
20 Gist	20	20	41.43±4.4	39.53±4.0
40 Gist	40	40	39.01±2.1	36.54±2.1
Gist_{arg}	4.59	4.59	48.78±1.7	47.85±1.2
HD-Gist	5.08	10.84	56.68±2.3	54.83±1.7
With Reconstruction Objective				
2 Gist	2	2	37.85±1.9	32.57±2.9
5 Gist	5	5	38.15±2.3	36.42±3.5
10 Gist	10	10	46.47±2.5	41.35±3.0
20 Gist	20	20	42.65±0.6	37.50±2.4
40 Gist	40	40	42.79±2.0	41.52±0.7
Gist_{arg}	4.59	4.59	51.34±0.3	48.68±0.4
HD-Gist	5.08	10.84	**71.22**±3.0	**69.24**±2.0

Table 1: Joint-goal accuracy on SGD (Rastogi et al., 2020) and SGD-X/v5 (Lee et al., 2022) test sets. All models are finetuned from a LLaMA 7B (Touvron et al., 2023) model. We report the results of static gist model (Mu et al., 2023) with up to 40 gist tokens. The best results from a model using the compressed API documentation are in bold. We report the mean and standard deviation across three random seeds.

Models	API Doc Tokens In Attention	API Doc Tokens In Memory	Accuracy
LLaMA	551.75	551.75	84.38
With Reconstruction Objective			
2 Gist	2	2	33.51
5 Gist	5	2	36.42
10 Gist	10	10	45.14
20 Gist	20	20	33.77
40 Gist	40	40	35.78
HD-Gist	4.15	12	**55.64**

Table 2: Accuracy of predicting the API and the model card on the APIBench (Patil et al., 2023) evaluation set.

5.2 Results on the APIBench Dataset

Next, we discuss the results on the APIBench (Patil et al., 2023) dataset. As shown in Table 2, the proposed HD-Gist model achieves a higher accuracy (55.65%) than all static gist-token models (45.14%). However, the gap (29%) to the LLaMA baseline with uncompressed API doc (84.38%) is much larger than it is on the SGD datasets. We believe this is because the documentation (e.g., descriptions of AI models) in APIBench is much longer than the documentation in SGD, which is demonstrated by the average number of tokens in the uncompressed API documentation (108.94 VS 551.75). In terms of the average compression ratio (original token to gist token ratio), one $\text{Gist}_{\text{value}}$ token of the lowest hierarchy only needs to encode a single value (e.g., "everyday" in Fig. 3) in SGD, while a same token of the lowest hierarchy is expected to encode the description of a model card (61.3 tokens on average) in APIBench. Given these difficulties, our proposed HD-Gist method still achieves decent performance gain, which demonstrates the generalizability of our method and intuition. The noticeable gap between the model with a full context raises another research question: when the lowest hierarchy of the input is still very long, it is necessary to either increase the capacity of the gist compression (more than 1 gist tokens) or introduce a finer-grained hierarchy (e.g., sentence) in the compression. We leave the exploration of this question to future work.

5.3 Error Analysis

We break down 5 different types of errors that models make on SGD and count the percentage of validation examples where the model makes a specific error. We divide the 5 errors into two categories: argument error and value error. Argument error is when a model (I) misses an argument that is in the ground-truth API call, (II) predicts an extra argument from the documentation but is not in the ground-truth API call, or (III) hallucinates an argument that is not even in the documentation.

The second category, value error (IV), is when a model predicts the wrong value for a categorical argument, or (V) it predicts the wrong value for a regular argument. For Type IV error, the model could predict the wrong category (e.g., "`s3.1` instead of `s3.2`", or predict a value instead of the desired index. We show the results in Table 3. We can observe that all static gist models (row 2-4) as well as the Gist_{arg} model predict the wrong value for a categorical argument (Type IV error) in more than 44% of the examples. The addition of the dynamic $\text{Gist}_{\text{value}}$ tokens significantly reduces the type IV error rate to 17.1% and adding reconstruction loss further reduces it to only 2.4%. This evidence corroborates our argument that the model can utilize more fine-grained information about an argument (e.g., its acceptable values) from the dynamic $\text{Gist}_{\text{value}}$ gist tokens.

5.4 Study on the Reconstruction Frequency

We then conduct a study on the percentage of training examples with the reconstruction loss. The results are shown in Table 4. On SGD-X, HD-Gist model achieves the best performance when we add the reconstruction loss in 10% of the training ex-

Models	Argument Error (%)			Value Error (%)	
	Miss (I)	Extra (II)	Hallu. (III)	Categorical (IV)	Regular (V)
LLaMA 7B	5.5	2.5	0.02	0.1	6.7
Append 2 Gist	11.3	8.5	1.1	47.0	15.8
Append 5 Gist	14.9	8.2	2.2	46.2	13.6
Append 10 Gist	9.9	3.2	1.3	46.6	7.4
Gist_{arg} Only	7.5	2.5	0.9	44.9	**6.9**
HD-Gist	10.5	2.8	0.3	17.1	7.2
+Reconstruction	6.7	**0.0**	2.7	**2.4**	7.2

Table 3: The absolute percentage of different errors made by different models on SGD validation set.

Rec. Ratio	HD-Gist		10 Gist	
	SGD	SGD-X	SGD	SGD-X
0.0	70.47±3.2	60.38±2.8	45.36±0.4	36.18±1.6
0.1	87.81±2.4	**83.37**±1.0	56.78±1.3	43.47±0.4
0.3	85.21±4.9	77.83±1.6	**70.62**±2.0	**55.84**±3.4
0.5	87.98±1.0	75.36±2.3	63.52±3.2	47.34±0.7
0.9	87.36±2.0	70.61±2.8	49.23±6.9	37.48±5.1
1.0	**89.98**±0.8	72.94±4.2	61.87±3.5	43.32±9.6

Table 4: Joint-goal accuracy (average and standard deviation over 3 seeds) of the models trained with different ratios of examples with reconstruction. We report the model with 10 static gist tokens and the model with HD-Gist, evaluated on SGD and SGD-X validation sets.

Caching Strategy	Time (ms)	Compute (GFLOPS)	Memory (GB)
None	743.4	5788.7	26.5
API Doc	727.6	4079.1	21.8
Static Gist Caching			
2 Gist	704.9	4059.0	17.6
5 Gist	706.0	4059.5	17.7
10 Gist	711.6	4060.3	17.8
20 Gist	710.7	4061.9	18.1
40 Gist	710.0	4065.1	18.8
Dynamic Gist Caching			
HD-Gist	705.7	4060.6	17.9

Table 5: Efficiency of different caching methods, evaluated on 100 SGD validation examples. We report the average CUDA time (millisecond), computation (giga-FLOPS), and memory usage (gigabyte) for generating the ground-truth API call.

amples, while the static gist model achieves the best performance with 30% training examples with reconstruction. Reconstructing in more or less examples also achieves improvements on the baseline with no reconstruction.

6 Efficiency Evaluation

6.1 Benchmarking Setup

In this section, we compare the efficiency of the HD-Gist model to the static gist-token model as well as the baseline with no prompt compression. We aim to answer one important question: does our proposed method still maintain the efficiency of the static gist-token model in terms of its compute, memory, and storage requirements? To answer this question, we compare the compute requirements (CUDA wall time, FLOPs) and memory usage during inference using different models and strategies to cache the API documentation:

- **No Caching**. We just encode the API documentation from scratch for every example.
- **API Doc Caching**. We cache the activations of the full API documentation (keys and values for all layers). This is the KV caching commonly used in the inference of a decoder-only Transformer (Pope et al., 2023).
- **Static Gist Caching** (Mu et al., 2023) compresses the API documentation into N gist tokens, and caches their activations.
- **Dynamic Gist Caching** compresses the API documentation into the proposed HD-Gist tokens, and caches their activations as well as a dictionary that maps a categorical argument's name (e.g., "`s3:`") to an attention mask that unmasks its $\text{Gist}_{\text{value}}$ tokens.

We benchmark the prediction step that generates an entire output instead of a single forward pass (at the first decoding step) as is done in Mu et al. (2023). This is because we want to take into account the extra time to switch between the general attention mask that only unmasks Gist_{arg} and targeted masks that additionally unmask a set of $\text{Gist}_{\text{value}}$ for a categorical argument. Since we aim to benchmark different models (LLaMA 7B with no compression, static gist-token model, and our proposed model), and each model may generate an output of different lengths, we benchmark these models for generating the same, ground-truth API call instead of actually decoding them. This enables us to make a fair comparison between the

efficiency of these models. We benchmark on a single NVIDIA A100 40GB and report the GPU time, compute and memory usage.

6.2 Benchmarking Results

Table 5 shows the results of profiling an entire prediction step with PyTorch (Paszke et al., 2019) 2.0, averaged across 100 random validation examples. First, we note that all caching methods achieve significant speedup, less compute and memory than "No Caching" that encodes the API documentation from scratch for every example. This demonstrates the efficiency of caching and reusing the API documentation's encodings.

Second, we observe that all static gist caching as well as the proposed dynamic gist caching only obtains a small speedup and reduction of compute compared to the "API Doc Caching". A similar trend is also observed in Mu et al. (2023) and it is because the FLOPs required for a Transformer forward pass is dominated by encoding the newly generated token (e.g., passing it through feed-forward layers), which is unchanged across all caching strategies, rather than computing the self-attention weights with the cached key-values. Although the improvements in speed are limited, using "Dynamic Gist caching" reduces memory usage by 17.9%. As is shown in Table 1, caching the entire API documentation requires caching the activations of 108.94 tokens on average, while the dynamic gist method only requires caching the activations of 10.84 gist tokens on average.

7 Discussion

In this section, we discuss HD-Gist's strong performance that even beats LLaMA with uncompressed API in SGD-X, and its potential of generalizing to compress any APIs and free text.

Compression as Regularization. One unexpected, but interesting finding in this work is that LLaMA with HD-Gist-compressed documentation outperforms LLaMA with uncompressed documentation in SGD-X (Table 1), whose arguments' names and descriptions are paraphrased by human annotators. The opposite is observed in the original SGD test set whose arguments' names and descriptions follow the same annotation as the training set. We believe this is because, after finetuning, LLaMA with uncompressed documentation overfits to the training examples. Therefore, when the arguments are paraphrased in the test set, the model is still predicting based on its memory of training APIs. LLaMA with HD-Gist-compressed documentation, on the other hand, is exposed to a minimum but sufficient amount of information about the API through HD-Gist tokens during training. Thus it is more robust to test-time variations in APIs and generalizes better according to the information bottleneck theory (Tishby and Zaslavsky, 2015).

Generalizing to Compress any APIs. In the real world, API documentations mostly follow a similar hierarchical structure: starting with a coarse-grained API description, then a list of arguments and their descriptions, and further fine-grained descriptions of acceptable values for categorical arguments. Therefore, for any API documentations, we can append a $\text{Gist}_{\text{value}}$ token after the description of a value, and append an Gist_{arg} token after the description of an argument. If the model needs to chain multiple API calls in the same expression, we can also append an API-level Gist token after an API documentation and hence include multiple APIs in the context. This allows HD-Gist to be generalized to compress any APIs.

Generalizing to Compress Free Text. We argue that HD-Gist can also be applied to compress free text where we have the ground-truth label on which part of the text the model should be attending. For example, in Multi-Hop Question Answering (Yang et al., 2018) with a long context containing multiple paragraphs, we know the golden paragraphs that contain the intermediate and final answers. Therefore we can add paragraph-level and sentence-level gist tokens to the context. The model only attends to paragraph-level gist tokens for the best efficiency, and then unmasks the sentence-level gist tokens once it predicts to use a certain paragraph in a chain-of-thought reasoning step.

8 Conclusion

In this work, we propose to compress API documentation into a few sets of hierarchical and dynamic gist tokens. We enable the model to unmask value-level gist tokens to zoom into more details of a categorical argument. We further present a reconstruction objective that improves the compressed gist representation. Empirical results on multiple datasets demonstrate the significant improvement upon a single set of static gist tokens without sacrificing the speed or incrementing FLOPs.

9 Limitations

Generalization to compress other texts. In this work, we propose to hierarchically compress an API documentation into a set of Gist_{arg} tokens and a sets of $\text{Gist}_{\text{value}}$ tokens. Each Gist_{arg} token is appended after the description of an argument and coarsely encodes this argument, while $\text{Gist}_{\text{value}}$ token is appended after an acceptable value of a categorical argument and finely encodes this specific value. The model can automatically zoom into/out of information about an argument of interest by dynamically unmasking and remasking its $\text{Gist}_{\text{value}}$ tokens. The decision of when to unmask and remask the $\text{Gist}_{\text{value}}$ tokens of a categorical argument and which argument's $\text{Gist}_{\text{value}}$ tokens to unmask is solely based on the partially generated API call (output). For example, when the partial output ends with a categorical argument "`a3: `", we unmask the $\text{Gist}_{\text{value}}$ tokens after every possible value of `a3` (e.g., "`[a3.0:` 1 $\text{Gist}_{\text{value}}$, `a3.1:` 0 $\text{Gist}_{\text{value}}$]"). After the model finishes predicting the value ("`a3=a3.0,`"), we mask these $\text{Gist}_{\text{value}}$ tokens again. Therefore, our method can be applied to compress any API documentation (e.g., python, pytorch, etc.) that has a naturally hierarchical structure.[5] As long as the API call output refers to the argument name as it is in the documentation, which is true in almost all programming languages, the model can automatically decide when to unmask the $\text{Gist}_{\text{value}}$ tokens of which argument.

One future direction is to extend the proposed hierarchical gist to compress unstructured, long context. To achieve this, one can explore some natural hierarchy (article, paragraph, sentence) within unstructured text and define and place gist tokens of different hierarchies. However, the output of a general prompt does not include signal tokens (e.g., argument names in API) that can be used to match to a component within the prompt hierarchy. Therefore, a crucial challenge is to let the model decide which paragraphs/sentences are relevant to the current decoding step and hence unmask the corresponding gist tokens. A potential solution is to quantify the "importance" of each paragraph/sentence within an article/paragraph using the attention weights on the paragraph/sentence gist tokens. We can then unmask the gist tokens in the most "important" paragraph/sentence to the current step.

Pretraining with compressed context. Another potentially impactful direction is to incorporate the gist compression into the pretraining language modeling objective, instead of finetuning a pretrained model to compress the context as is done in this work and Mu et al. (2023). This can significantly increase the length of the context window, which is a crucial factor in the pretraining of LLMs as the memory usage scales quadratically with the context length. For example, the longest context length of LLaMA is 2048 and further context longer than 2048 has to be truncated. Assume each paragraph in a corpus has 100 tokens. We can train the LLaMA model to attend to a token that is 409600 ahead by compressing each paragraph into a gist token.

[5] Each API has multiple required and optional arguments, among which some arguments are categorical and have a finite set of acceptable values.

Ethical Considerations

In this work, we finetune a LLaMA 7B (Touvron et al., 2023) model to compress the API documentation and then predict the API call based on the user's request. All training data are open-sourced, and hence do not contain any private or sensitive information. Nonetheless, previous work has shown that models trained with these large corpora can sometimes generate outputs that are toxic (Dinan et al., 2019) or reflect gender bias (Dinan et al., 2020) that might be offensive to certain users. As we are solely interested in having the model to predict the API call, we do not assess the toxicity or faithfulness of the model in generating free-form responses. Therefore, the model presented is only intended to predict an API call from the API documentation and user's request. It is not intended to act as a chat agent on its own and we do not recommend prompting this model to generate free-form content.

References

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. 2023. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3829–3846, Singapore. Association for Computational Linguistics.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Emily Dinan, Angela Fan, Adina Williams, Jack Urbanek, Douwe Kiela, and Jason Weston. 2020. Queens are powerful too: Mitigating gender bias in dialogue generation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8173–8188, Online. Association for Computational Linguistics.

Emily Dinan, Samuel Humeau, Bharath Chintagunta, and Jason Weston. 2019. Build it break it fix it for dialogue safety: Robustness from adversarial human attack. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4537–4546, Hong Kong, China. Association for Computational Linguistics.

Nouha Dziri, Sivan Milton, Mo Yu, Osmar Zaiane, and Siva Reddy. 2022. On the origin of hallucinations in conversational models: Is it the datasets or the models? In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5271–5285, Seattle, United States. Association for Computational Linguistics.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Tao Ge, Jing Hu, Xun Wang, Si-Qing Chen, and Furu Wei. 2023. In-context autoencoder for context compression in a large language model. *arXiv preprint arXiv:2307.06945*.

Joy He-Yueya, Gabriel Poesia, Rose E Wang, and Noah D Goodman. 2023. Solving math word problems by combining language models with symbolic solvers. *arXiv preprint arXiv:2304.09102*.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. *arXiv preprint arXiv:2310.06839*.

Hoyoun Jung and Kyung-Joong Kim. 2023. Discrete prompt compression with reinforcement learning. *arXiv preprint arXiv:2308.08758*.

Mojtaba Komeili, Kurt Shuster, and Jason Weston. 2022. Internet-augmented dialogue generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8460–8478, Dublin, Ireland. Association for Computational Linguistics.

Angeliki Lazaridou, Elena Gribovskaya, Wojciech Stokowiec, and Nikolai Grigorev. 2022. Internet-augmented language models through few-shot prompting for open-domain question answering. *arXiv preprint arXiv:2203.05115*.

Harrison Lee, Raghav Gupta, Abhinav Rastogi, Yuan Cao, Bin Zhang, and Yonghui Wu. 2022. Sgd-x: A benchmark for robust generalization in schema-guided dialogue systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10938–10946.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. 2023. Compressing context to enhance inference efficiency of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6342–6353, Singapore. Association for Computational Linguistics.

Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. 2023. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *arXiv preprint arXiv:2303.16434*.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. 2023. Learning to compress prompts with gist tokens.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. 2020. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. 2020. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 8689–8696.

Siyu Ren, Qi Jia, and Kenny Zhu. 2023. Context compression for auto-regressive transformers with sentinel tokens. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12860–12867, Singapore. Association for Computational Linguistics.

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.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. 2022. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*.

Naftali Tishby and Noga Zaslavsky. 2015. Deep learning and the information bottleneck principle. In *2015 ieee information theory workshop (itw)*, pages 1–5. IEEE.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. 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.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Peitian Zhang, Zheng Liu, Shitao Xiao, Ninglu Shao, Qiwei Ye, and Zhicheng Dou. 2024. Soaring from 4k to 400k: Extending llm's context with activation beacon. *arXiv preprint arXiv:2401.03462*.

Jeffrey Zhao, Raghav Gupta, Yuan Cao, Dian Yu, Mingqiu Wang, Harrison Lee, Abhinav Rastogi, Izhak Shafran, and Yonghui Wu. 2022. Description-driven task-oriented dialog modeling. *arXiv preprint arXiv:2201.08904*.

Appendix

A Method

In Fig. 3, we show an example with the modified attention mask from the model with HD-Gist tokens that is also supervised to reconstruct the API documentation. During reconstruction, the model can only attend to the Gist_{arg} tokens. When it starts reconstructing a categorical argument, we unmask the $\text{Gist}_{\text{value}}$ tokens associated with that argument, so that the model can learn to encode the fine-grained information (the list of all acceptable values) of the categorical argument into these $\text{Gist}_{\text{value}}$ tokens.

B Experimental Setup

B.1 Datasets

SGD (Schema-Guided Dialogue) (Rastogi et al., 2020) dataset challenges models to perform dialogue state tracking (DST) by following a schema. It has 143,346 training examples, 21,026 validation examples, and 36,129 test examples. The schema consists of multiple services (e.g., Hotel), where each service includes multiple intents (e.g., `FindHotel`) that can be invoked to fulfill a user's request. Each intent is like an API and takes a number of arguments (e.g., location of hotel), including categorical arguments (e.g., "Number of guests per room") that have a list of acceptable values ("[1, 2, 3]"). We convert the original DST task into an API prediction task by (1) discarding arguments that are not used by the currently active intent from the output[6] and (2) giving the model documentation of intent instead of the whole service. In both training and test, we also include intermediate turns where the user has not yet provided all arguments' values. In these turns, we ask the model to generate a partial API call with only those arguments mentioned by the user so far. This increases the size of the training set at zero cost by utilizing the supervision from a dialogue state tracking dataset with labels of active arguments after every user's turn. The dataset does not include any information that would leak the unique identity of individuals.

SGD-X (Lee et al., 2022) is created from SGD by asking human annotators to paraphrase the original arguments' names and descriptions into se-

[6]For example, the DST task tracks the argument "location of hotel" specified by the user for the previous intent "`FindHotel`" but is not relevant to the current intent "`BookHotel`".

mantically similar yet stylistically diverse variants. For example, the argument "`RequestPayment`: Request payment from someone" is rewritten as "`TransferRequest`: Ask for a money transfer from a contact". SGD-X further evaluates models' robustness to linguistic variations in API documentation. We use the SGD-X/v5, which is the version with the most variation, as the extra test set to evaluate models trained on the original SGD training set.

APIBench (Patil et al., 2023) is a dataset consisting of APIs from HuggingFace, TorchHub, and TensorHub as well as 10 user question prompts generated from Self-Instruct (Wang et al., 2023). The documentation in APIBench only include the 3 API that construct a pretrained model (e.g., `AutoModel.frompretrained` in pytorch) and the acceptable values (model card's url and description) of the first argument. There is no documentation on how to further use the constructed model to process inputs provided by users. Because the sole purpose of this work is to train and evaluate models to follow a compressed documentation, we discard the undocumented parts of API calls and only ask the model to predict (1) one out of three available APIs (HuggingFace, TorchHub, TensorHub), and (2) the value (model card's url) of the first and only argument of an API. Therefore, we insert an API-level gist token after every API, insert a value-level gist token after the description of every model card, and omit the argument-level gist token since there is only one argument to predict.

We further observe that some user request does not specify which API they want to use, and all three APIs have at least one AI model that suffices the request. To eliminate the ambiguity, we add a prompt "I want to use TensorHub/TorchHub/Huggingface" to the user's request. For each example, we sample 2 distracting model cards from the different categories of same API and 3 distracting model cards from the other two APIs. For example, if the ground-truth model card is a sentiment analysis model from TorchHub, we will not sample distracting model cards from the sentiment analysis category of TorchHub. However, we might sample a sentiment analysis model from Huggingface or TensorHub as a distractor. We repeat this sampling process 5 times per example to create 5 training instances with different distracting model cards. The resulting dataset has 48,750 training examples and 1,143 evaluation examples.

Models	API Doc Tokens in attention	API Doc Tokens in memory	Accuracy SGD	Accuracy SGD-X
LLaMA	109.43	109.43	90.09	73.03
Without Reconstruction Objective				
2 Gist	2	2	41.98	32.02
5 Gist	5	5	42.29	33.70
10 Gist	10	10	45.48	38.00
20 Gist	20	20	41.42	33.41
40 Gist	40	40	42.85	34.24
Gist_{arg}	4.09	4.09	48.46	43.00
HD-Gist	4.96	10.62	64.46	54.75
With Reconstruction Objective				
2 Gist	2	2	54.92	41.01
5 Gist	5	5	60.68	48.26
10 Gist	10	10	73.48	60.57
20 Gist	20	20	65.02	46.70
40 Gist	40	40	72.57	53.98
Gist_{arg}	4.09	4.09	71.38	58.90
HD-Gist	4.96	10.62	**88.37**	**84.30**

Table 6: Joint-goal accuracy of single models on SGD (Rastogi et al., 2020) and SGD-X/v5 (Lee et al., 2022) validation sets. The best results from a model using the compressed API documentation are in bold.

B.2 Training Details

We finetune every model on 8 NVIDIA A100 40GB GPUs for a single epoch, which takes around 16-18 hours to finish.

B.3 Evaluation Metrics

Following Rastogi et al. (2020), we evaluate models on SGD and SGD-X using joint-goal accuracy. For arguments that are both in the ground truth and the predicted API call, we calculate the exact-match scores for values of categorical arguments, and calculate fuzzy soft-matching scores[7] for other arguments. For example, predicting "New York" while the ground-truth is "New York City" results in a fuzzy-matching score of 0.76. A matching score of 0 is assigned for both of the following errors: (i) arguments that in the ground truth but missed in the prediction, (ii) arguments in the prediction but not the ground truth. The joint-goal accuracy is the product of matching scores of all arguments in the API documentation. For APIBench that has no non-categorical argument, we use the exact-match accuracy only.

C Extra Results

In Table 6, we report the models' joint-goal accuracy on the SGD and SGD-X validation sets. The results are evaluated on the single model trained with seed 42.

[7] https://pypi.org/project/fuzzywuzzy/