Concise and Precise Context Compression for Tool-Using Language Models

Yang Xu[1*], **Yunlong Feng**[1], **Honglin Mu**[1], **Yutai Hou**[2], **Yitong Li**[2], **Xinghao Wang**[2], **Wanjun Zhong**[2], **Zhongyang Li**[2], **Dandan Tu**[2], **Qingfu Zhu**[1†], **Min Zhang**[1], **Wanxiang Che**[1]
[1]Harbin Institute of Technology
[2]Huawei Technologies Co., Ltd
{yxu,qfzhu}@ir.hit.edu.cn

Abstract

Through reading the documentation in the context, tool-using language models can dynamically extend their capability using external tools. The cost is that we have to input lengthy documentation every time the model needs to use the tool, occupying the input window as well as slowing down the decoding process. Given the progress in general-purpose compression, soft context compression is a suitable approach to alleviate the problem. However, when compressing tool documentation, existing methods suffer from the weaknesses of key information loss (specifically, tool/parameter name errors) and difficulty in adjusting the length of compressed sequences based on documentation lengths. To address these problems, we propose two strategies for compressing tool documentation into concise and precise summary sequences for tool-using language models. 1) Selective compression strategy mitigates key information loss by deliberately retaining key information as raw text tokens. 2) Block compression strategy involves dividing tool documentation into short chunks and then employing a fixed-length compression model to achieve variable-length compression. This strategy facilitates the flexible adjustment of the compression ratio. Results on API-Bank and APIBench show that our approach reaches a performance comparable to the upper-bound baseline under up to 16x compression ratio.

1 Introduction

The advent of tool-using language models represents a significant extension of the capability of language models, including but not limited to interacting with the Internet through the web browser, retrieving knowledge from an extended database, or even driving other models or devices (Nakano et al., 2021; OpenAI, 2023; Patil et al., 2023; Cheng et al., 2023). As a common practice, a tool-using language model can be modeled as a function calling model built upon the natural language interface (Patil et al., 2023; Li et al., 2023a; Qin et al., 2023; Tang et al., 2023). Specifically, the user inputs the tool documentation along with the instruction, then the model generates a structured function call defined by the documentation.

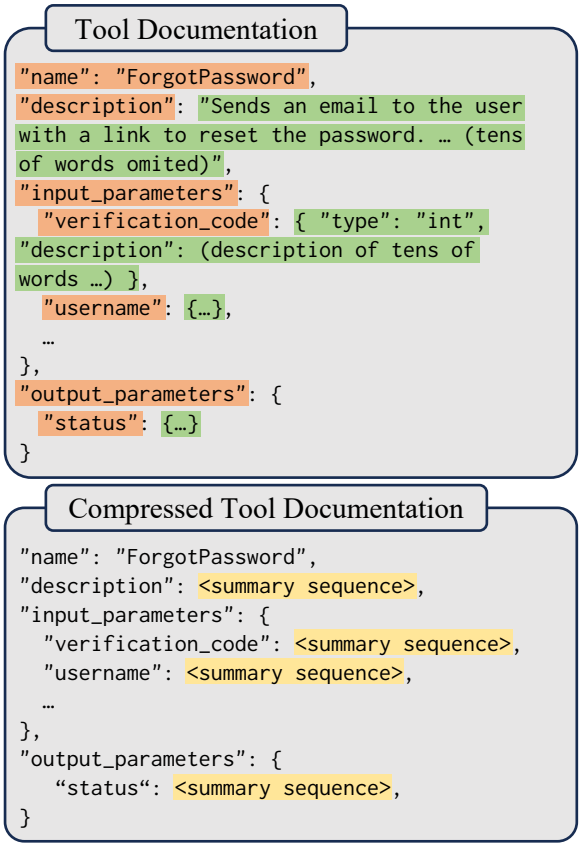

Figure 1: An example of tool documentation compression. Compared with the key information (red), the other content (green) is more verbose and suitable for compression into summary sequences (yellow).

Nevertheless, the tool documentation can be lengthy, occupying a fixed-size input window as long as we use the tool. To fully unleash the capability of tool-using language models, the tool

[*]work done during internship at Huawei
[†]corresponding author

Findings of the Association for Computational Linguistics ACL 2024, pages 16430–16441

documentation has to include a detailed description of the functionality and input/output format, which can cost hundreds of tokens per tool. Moreover, in the cases of multiple tools, like the combination of tools or online top-k retrieval of suitable tools, documentation can easily accumulate to over a thousand tokens.

Fortunately, as shown in Figure 1, tool documentation shows the potential to be compressed given its nature. First, the model only needs to understand the tool's functionality no matter how it is described, which means that we can use a summarized version for most parts of the lengthy documentation. In contrast, the parts containing key information, such as names and formats marked red in Figure 1, must be kept as is, or the generated function call can easily fail. In this work, we regard the key information as names of tools and parameters in the tool documentation. Second, the documentation is fixed once the tool is deployed, enabling a one-time compression as preprocessing.

In this work, we propose to compress the tool documentation in the context into concise and precise summary sequences. Soft context compression approaches (Ge et al., 2023; Chevalier et al., 2023; Bulatov et al., 2022; Wang and Xiao, 2024) offer a sound general-purpose compression framework. However, these approaches suffer from uncontrollable compression loss and lack of support for setting compression ratio when compressing tool documentation. Therefore, we propose two strategies to improve context compression for tool-using language models.

The first strategy is selective compression, which mitigates compression loss on key information, i.e., names of tools and parameters. We propose the construction of summary sequences in an interleaved format of compressed and uncompressed sub-sequences, wherein key information is preserved as original text tokens. With less concern about losing key information, it is less challenging to condense summary sequences to a shorter length.

The second strategy is block compression which compresses documentation according to a fixed compression ratio instead of a fixed summary sequence length. To achieve a controllable target compression ratio, we split the documentation into blocks according to the preset compression ratio, and then perform fixed-length compression separately. In this manner, we no longer need to be concerned about length generalization or the wastage of summary tokens when dealing with documentation of a wide variety of lengths.

With these strategies, our approach starts from a pre-trained model, and then applies continual pre-training and fine-tuning pipeline to train the model to generate and leverage summary sequences.

We evaluate our approach on two tool-using benchmarks: API-Bank (Li et al., 2023a) and APIBench (Patil et al., 2023). Results show that under the compression ratio of up to 16x, our approach reaches comparable performance with the upper-bound baseline without compression.

We also explore the influence of our two proposed strategies on the same benchmarks. Results show that selective compression significantly mitigates compression loss of key information, enabling a higher compression ratio. Compared to overall compression, our block compression brings no additional compression loss.

Our primary contributions are as follows:

- We introduce concise and precise context compression for tool-using language models, with strategies for minimizing key information loss under variable compression ratio.
- Our approach on two tool-using benchmarks demonstrates negligible performance loss under up to 16x compression ratio.
- We explore different combinations of training objectives and compression strategies, and provide a recipe for context compression training for tool-using language models.

2 Related Work

Tool-Using Language Models In terms of flexibility, the tool-using capability of language models can be broadly categorized into two types: supporting limited pre-defined tools as built-in features, or enabling arbitrary tools by reading corresponding documentation in the context. The first involves integrating predefined tools like search engines, calculators, and Python interpreters (Cobbe et al., 2021; Nakano et al., 2021; Komeili et al., 2022; Thoppilan et al., 2022; Gao et al., 2022; Huang et al., 2022; Yao et al., 2022; Cheng et al., 2023; Schick et al., 2023). The second more dynamically trains LMs to utilize tools by reading their documentation, expanding their application scope. This approach is exemplified by ChatGPT plugins (OpenAI, 2023) and further developments in open-source LMs (Yang et al., 2023; Xu et al.,

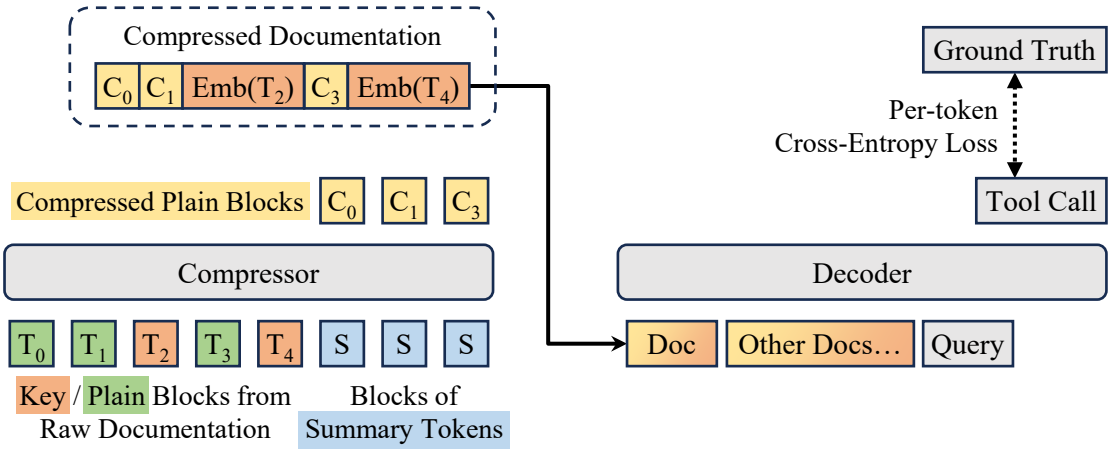

Figure 2: Overview of our method for tool documentation compression. When compressing a tool's documentation, we cut out the key information as key blocks (red) and chunk the rest into plain blocks (green). We use the concatenation of key blocks and compressed plain blocks (yellow) as the compressed documentation. We supervise the decoder output conditioned on compressed documentation to train the compressor and the decoder end-to-end.

2023; Patil et al., 2023; Li et al., 2023a; Qin et al., 2023; Tang et al., 2023), focusing on API usage and creating benchmarks for tool-using LMs.

Our study focuses on the second approach, where LMs generate structured function calls given tool documentation and instructions, illustrating practical application and flexibility in tool utilization.

Context Compression Context compression in large language models (LLMs) is pivotal for enhancing their efficiency. This research domain is mainly divided into soft summary and hard token pruning methods.

Soft summary methods, requiring pre-training or fine-tuning pre-trained language models as the compressor, yield high compression rates through condensed context representations. Prominent research in this area (Bulatov et al., 2022; Ge et al., 2023; Wang and Xiao, 2024) primarily utilizes memory tokens for context compression, yet often lacks focus on adjustable compression ratios.

Conversely, hard token pruning, exemplified by Li et al. (2023b); Jiang et al. (2023a,b), entails the elimination of non-essential text by identifying and truncating non-informative sections. Specifically, they measure the amount of information with self-information or the perplexity from language models. Unlike soft summary methods, these methods leverage pure text protocol, thus easy to apply to black box models.

Our contribution enhances soft summary methods by integrating tool-use-specific strategies in LLMs and prioritizing controllable compression ratios. This approach, differing from works like Chevalier et al. (2023); Ge et al. (2023), provides a customized solution for the specialized task in tool-using language models.

3 Method

Our approach uses two strategies to improve the basic soft context compression for tool-using language models. In this paper, we name the language model which generates the output as the decoder, and refer to the model which generates the compressed sequence as the compressor. Although the decoder and the compressor are the same model in our work, we differentiate them from each other in our description for clear logic. We start with basic context compression, then introduce our two strategies, and finally describe our approach which integrates all of them.

3.1 Basic Soft Context Compression

As our starting point, basic soft context compression approaches aim to compress an arbitrary token sequence to a fixed length sequence of soft tokens (Bulatov et al., 2022; Chevalier et al., 2023), which we call overall compression.

To achieve this goal, a two-pass pipeline is performed to compress and then leverage the soft token sequence.

The first pass compresses the token sequence into a soft token sequence. Assuming we have a token sequence $T = [t_0, t_1, ..., t_{L_T-1}]$ to compress to L_S soft tokens, we append a special summary token sequence $S = [s_0, s_1, ..., s_{L_S-1}]$ to T and

input them to the compressor, obtaining the output hidden state

$$H = \text{Compressor}(\text{Emb}(T) \| \text{Emb}(S)).$$

Then the hidden states $C = H[L_T, L_T + L_S)$ correspond to S are used as the compressed soft token sequence, thus we have

$$\text{Compressed}(T) = H[L_T, L_T + L_S).$$

The second pass leverages the soft token sequence as a soft prompt. When generating the output for a context T' conditioned on the compressed sequence T, C is used as the alternate of $\text{Emb}(T)$, thus we have the decoder output

$$H' = \text{Decoder}(C \| \text{Emb}(T')).$$

To train the compressor and decoder models, the language modeling training objective is applied to H' as an indirect supervision since we cannot obtain the gold answer of soft tokens.

3.2 Selective Compression Strategy

The basic compression approaches have to face the challenge of compression loss. However, the compression loss is difficult to control. The higher the compression ratio, the greater the compression loss. Even if trained with reconstruction loss, the decoder can make mistakes when recovering words from compressed tokens, especially on rare words, just like human (Ge et al., 2023).

This feature is harmful to tool-using language models, since key information loss such as misspelled names of tools or parameters will directly lead to failure. Therefore, we propose the selective compression strategy as a more controllable approach to keep the key information despite the compression ratio, where the key information retains raw text tokens.

Given a tool documentation as the context T to be compressed, we split it into disjoint subsequences $T_0, T_1, ..., T_{N_{\text{subseq}}-1}$ whose union is T. Each T_i is rather a key information sub-sequence (e.g., name of a parameter) or a sub-sequence that can be compressed (e.g., functionality description of the tool). Following the notations in basic soft context compression, we have the compressed token sequence

$$C = \|_i C_i \text{ , where}$$

$$C_i = \begin{cases} \text{Emb}(T_i) & T_i \text{ is key information} \\ \text{Compressed}(T_i) & \text{otherwise} \end{cases}.$$

3.3 Block Compression Strategy

A notable challenge in context compression for tool-using language models is the variability in documentation length. The basic compression methods typically compress the documentation into a uniform sequence of fixed length, denoted as L_S. This approach has limitations: for lengthy documentation, setting a small L_S leads to a significant loss of information, adversely affecting performance. Conversely, a large L_S hinders the effective compression of shorter documents, thereby reducing the compression ratio. Additionally, applying selective compression strategies, which divide the documentation into sub-sequences of varying lengths, intensifies these issues.

We believe it is a better approach to compress tool documentation according to a preset compression ratio r. To realize this approach, we propose the block compression strategy to support the variable r with fixed L_S. The core idea is to chunk the sequence to be compressed into a variable number of blocks, each compressed to L_S soft tokens.

We chunk the sequence to compress T into $N_{\text{chunk}} = \lceil \frac{L_T}{r \times L_S} \rceil$ chunks $T_0, T_1, ..., T_{N_{\text{chunk}}-1}$. Concatenating the compressed version of these chunks, we obtain the final compressed sequence

$$C = \|_i \text{Compressed}(T_i).$$

Note that the last chunk is not always full, which will make the compressed sequence at most L_S soft tokens longer than expected. Therefore, L_S should be a small number.

3.4 Concise and Precise Context Compression

The aforementioned strategies offer a concise and precise approach to compress tool documentation for tool-using language models. As illustrated in Figure 2, our final method integrates both of them based on the basic soft context compression.

Combining Selective and Block Compression Strategies From the perspective of block compression, we can unify the two strategies in practice by regard key information sub-sequences as special blocks which we do not compress. Specifically, we first split the key information sub-sequences (i.e., the key blocks), then chunk the other sub-sequences into blocks (i.e., the plain blocks).

Next, we input the blocks into the compressor. To reach high efficiency and keep more context information, we compress all the blocks in one

documentation in parallel. As shown in the compression part of Figure 2, we append one block of summary tokens to the input sequence for each plain block, and obtain all the compressed blocks at once.

Finally, as described in the selective compression strategy, we concatenate all the blocks to form the compressed documentation, which is then used by the decoder.

Training Compressor and Decoder Following the basic soft context compression approaches (Chevalier et al., 2023; Ge et al., 2023), we initialize the compressor and the decoder with a pre-trained language model, then jointly train them. Actually, we use the same model as the compressor and the decoder, when we input summary tokens, it outputs compressed blocks, otherwise, it works as an ordinary language model. The compressor and decoder need pre-training to acquire the capability of using soft tokens. Different from existing approaches, our pre-training format changes due to the integration of our two strategies. Specifically, we randomly chunk pre-training data as key blocks and plain blocks, and perform the same parallel compression manner as shown in Figure 2.

The training objective is language modeling, thus we apply the per-token cross-entropy loss on the decoder output. With the gradient propagated by the compressed documentation, the loss can supervise the decoder as well as the compressor in an end-to-end manner.

In addition, Ge et al. (2023) propose to add an auxiliary loss of reconstructing the raw text T from the compressed soft token sequence C, which agrees to our goal of keeping key information, so we take this idea into account in our method, implementing a variant of our approach with reconstruction loss. In practice, we follow Ge et al. (2023) to use the trainable soft prompt to switch the decoder between ordinary mode and reconstruction mode.

4 Experiments

4.1 Basic Settings

To evaluate our approach, we conduct experiments to train models with context compression and test the performance on tool-using benchmarks. Our goal is to investigate the variations in model performance under different compression ratios and compression strategies.

Base Models In all of our experiments, we use the same base model as both the compressor and the decoder, and use LLaMA-7b (Touvron et al., 2023) to initialize all the base models. According to the manner of compression, we categorize base models into three cases:

- No compression: this case corresponds to the fine-tuned LLaMA without context compression, which we regard as the upper-bound approach because it is not affected by compression loss.
- Overall context compression: these models act as our baselines, performing overall compression in the manner of basic soft context compression approaches.
- Selective context compression: these models benefit from our proposed selective compression strategy and demonstrate the performance of our approach.

In these cases, block compression is always enabled so that we can track the variation of model performance with a controllable compression ratio.

Note that we refer to existing compression approaches such as RMT (Bulatov et al., 2022) and AutoCompressor (Chevalier et al., 2023) as basic compression, and the overall context compression case is equivalent to basic compression plus block compression. Therefore, we study the influence of block compression strategy through extra analysis experiments in Section 4.5 which provide fair comparison between our approach and these existing soft compression approaches.

Key Information for Selective Compression For general context, it is difficult to key information for selective compression. However, when we focus on the documentation of tools, it becomes evident that the information that models must precisely retrieve is the name. Therefore, we follow a simple but sound definition of key information as the names of tools and parameters.

Training Objectives We consider two training objectives in our experiments. The first is language modeling in the supervision of the decoder output. It is the basic objective for all the base models including the no-compression case. The second is the reconstruction objective, requiring the decoder to recover the raw text from compressed soft token sequences. Reconstruction is only for models with compression, acting as an auxiliary objective. When the reconstruction objective is on, we use the sum of reconstruction and the language modeling

loss as the final loss.

The reconstruction objective is from ICAE (Ge et al., 2023) which shares the same motivation of keeping the raw information and compatible with our approach. From this perspective, ICAE is parallel to our approach. Therefore, we study the effects of the reconstruction loss as long as it is possible. Specifically, switching the reconstruction objective on and off, each base model in the compression cases has two variants. We report and analysis results of both variants.

4.2 Pre-training Compression Models

Following existing soft compression approaches (Chevalier et al., 2023; Ge et al., 2023), we pre-train base models on general corpus at first instead of directly fine-tuning them on downstream tasks. The only exception is the upper-bound baseline, i.e., a fine-tuned LLaMA without context compression, for which pre-training is omitted because it does not need to acquire how to use soft tokens.

Dataset We use SlimPajama (Soboleva et al., 2023) as the pre-training dataset. SlimPajama is a deduplicated version of RedPajama (Computer, 2023), which is a community reproduction of the LLaMA (Touvron et al., 2023) pre-training dataset. To pre-train LLaMA with compression, we randomly sample data within 2k context length to construct a subset of 1B tokens. Then, we train all the models with compression on the same subset for one epoch.

Compression Manner We maintain the compression manner during pre-training consistent with the manner in practice. For overall compression models, we randomly select a prefix for each sample, with the length ranging from 0.5k to 1.5k, and then use the compressed prefix as input, with the remaining portions serving as output. For selective compression models, we randomly mark sub-sequences of the prefix as key information according to a random proportion ranging from 0 to 1 for each sample.

Given the heavy computation cost of pre-training, to ensemble support for variable compression ratio in one model, we assign a random compression ratio ranging from 1 to 16 to each sample, and always set the length of the summary sequence as 2.

Dataset	#Samples Train / Test	#Tools	Averaged Doc Length
API-Bank	6184 / 389	53	129
APIBench	15034 / 1785	1726	356

Table 1: Statistics of tool-using benchmarks where documentation lengths are counted by the LLaMA tokenizer.

Instruction-Tuning Some tool-using language models including the official baseline of API-Bank are fine-tuned on instruction-tuned models (Li et al., 2023a; Tang et al., 2023; Taori et al., 2023; Chiang et al., 2023). To unify the settings and make our base models prepared to efficiently adapt to downstream tasks, we follow them to instruction-tune the pre-trained compression models as the final base models. Specifically, we use the ShareGPT dataset released by OpenChat (Wang et al., 2023) as the instruction-tuning dataset. We trunk ShareGPT into a sequence length of 2k and train one epoch for all the models, with the other settings and training procedure the same as pre-training.

4.3 Tool-Using Benchmarks

We evaluate our approach on tool-using benchmarks API-Bank (Li et al., 2023a) and APIBench (Patil et al., 2023). Both of them can be modeled as a standard case in the decoder part of Figure 2, in which tool-using language models accept one or more tool documentation as well as the user query as input and then output the tool call.

API-Bank The dataset consists of multi-turn dialogues, where the user can ask the model to call external APIs. Each tool documentation is a JSON dictionary as exemplified in Figure 1. We use the `level1-api` subset of API-Bank in our experiments, where each of the user queries is a dialog history ending with an instruction to use a tool.

APIBench The dataset simulates a scene where an automated agent finds a suitable model on a platform (e.g., Hugging Face Hub) to fulfill the user's query. Therefore, the input contains a query and the model card of candidate models as the tool documentation, and the output is an API call to drive the model. However, the dataset is originally for testing retrieval-augmented tool-using language models. Specifically, only the top-ranked candidate model retrieved is provided to the decoder, and it cannot be guaranteed that the retrieval is correct, thus introducing the possibility of cascading errors.

Compression Ratio	API-Bank				APIBench			
	4	8	12	16	4	8	12	16
No compression	71.47				88.24			
Overall context compression	68.12	67.10	64.52	61.70	88.18	88.12	85.15	85.71
w/ Reconstruction loss	64.27	66.58	62.98	53.21	82.80	84.09	82.97	79.16
Selective context compression	**70.18**	**72.75**	**69.15**	**72.49**	**90.31**	89.58	87.79	**89.13**
w/ Reconstruction loss	69.41	67.35	68.64	69.67	88.52	**89.75**	**88.85**	88.29

Table 2: Accuracy on the test set of two tool-using benchmarks. The performance of selective compression is seldom affected by the compression ratio, however, the performance of overall compression noticeably decays as the compression ratio increases.

To weaken the impact of the retrieval module, we use the BM25 retrieval module in the official codebase to extend the number of candidates to up to 5, and make sure the correct answer is within. We shuffle the order of documentation to avoid potential position bias. The original dataset contains three subsets (i.e., `huggingface`, `tensorflowhub`, and `torchhub`), we use their union as a whole dataset.

Fine-tuning on Downstream Tasks We use the official training and test set for both of the datasets, whose statistics are shown in Table 1. As shown in Figure 2, we always compress different tool documentation separately, then concatenate all the compressed documentation before inputting them to the decoder. To ensure the consistency between training and testing, we train a separate model for each combination of the compression ratio, the base model, and the dataset. We train all the models on the corresponding dataset for two epochs at once.

Metric Both of the benchmarks use the accuracy of API calls as the metric. API-Bank provides a local sandbox to run the APIs, and check the running result to judge whether the API call is correct or not. APIBench checks the answer through AST matching, without running the API. We follow the official test approaches for both of the benchmarks.

4.4 Tool-Using Evaluation

Table 2 demonstrates the performance of three cases of base models. In general, selective context compression outperforms overall context compression, reaching similar or even higher performance compared to the no-compression case. Also, we find that adding reconstruction loss can be harmful to performance, and selective context compression can close the performance gap caused by reconstruction loss.

Comp. Ratio	API-Bank				APIBench			
	4	8	12	16	4	8	12	16
No comp.	21				101			
Overall comp.	39	31	41	51	106	107	128	129
w/ Rec. loss	53	41	45	93	178	168	185	209
Selective comp.	30	**21**	**33**	**21**	**83**	86	107	**85**
w/ Rec. loss	**24**	36	**33**	27	103	**85**	**99**	111

Table 3: Number of name errors corresponds to Table 2, lower is better.

In these two benchmarks, trends of performance according to the compression ratio differ. Table 1 shows that documentation in API-Bank is more concise than those in APIBench, thus intuitively harder to compress. Results show that the performance of overall context compression noticeably decays when the compression ratio becomes higher, which supports this intuition. On the other hand, only the weakest base model, which is overall context compression with reconstruction loss, shows obvious performance degradation on APIBench. This phenomenon suggests that APIBench is much easier than API-Bank, having the potential to keep satisfactory performance under a higher compression ratio.

Furthermore, we dive into the error cases to explore whether selective context compression keeps key information or not. Under our intuitive definition of key information as the tool names and parameter names, we count the number of error cases caused by name error, i.e., the model predicts a wrong name of APIs or parameters.

From results aggregated in Table 3, we find selective compression models make fewer mistakes in name errors. On APIBench, this phenomenon is more obvious. Although overall compression can reach comparable performance with the no compression baseline, the number of name errors grows with the compression ratio. In contrast, selective

Approach	Pre-training Strategy	Fine-tuning Strategy	API-Bank				APIBench			
			4×	8×	12×	16×	4×	8×	12×	16×
No compression	n/a	n/a	71.47				88.24			
Compression	Overall	Overall	68.12	67.10	64.52	61.70	88.18	88.12	85.15	85.71
	Selective	Overall	67.35	60.15	69.67	67.10	88.07	88.12	89.19	86.11
	Overall	Selective	**70.44**	69.15	**70.18**	68.38	85.49	88.63	**89.75**	**89.41**
	Selective	Selective	70.18	**72.75**	69.15	**72.49**	**90.31**	**89.58**	87.79	89.13
Compression w/ Reconstruction loss	Overall	Overall	64.27	66.58	62.98	53.21	82.80	84.09	82.97	79.16
	Selective	Overall	66.84	65.30	65.30	65.81	86.05	87.34	83.64	83.03
	Overall	Selective	68.89	**69.15**	66.84	65.04	87.00	87.96	85.49	85.94
	Selective	Selective	**69.41**	67.35	**68.64**	**69.67**	**88.52**	**89.75**	**88.85**	**88.29**

Table 4: Exploration on the effects of selective compression in different training stages. Compression models can adapt to selective compression with fine-tuning. The best combination is to consistently train with selective compression. 4× to 16× represents the compression ratios.

	API-Bank	APIBench
No compression	71.47	88.24
Compress each documentation separately		
Basic context compression	56.81	82.52
w/ Block context compression	**69.92**	**85.88**
Compress all documentation as a whole		
Basic context compression	51.41	80.39
w/ Block context compression	**64.52**	**81.85**

Table 5: When compressing documentation to 50 soft tokens, the proposed block compression greatly benefits existing basic compression approach, i.e., applying RMT or AutoCompressor directly.

compression keeps the number of name errors even lower than baseline, less affected by the compression ratio.

Another interesting finding is that the overall compression base model with reconstruction loss produces even more name errors, suggesting that without priors it is hard for the model to realize the importance of names.

4.5 Effects of Block Compression Strategy

Since supporting of controllable compression ratio relies on block compression, the methodology of observing the performance in different compression ratios does not work when studying block compression itself. To make a fair comparison between cases with and without block compression, we step back to the basic soft context compression. This section also plays the role of providing fair comparison between existing soft compression approaches, namely RMT (Bulatov et al., 2022) and AutoCompressor (Chevalier et al., 2023), and the proposed block comporession strategy.

Actually, the overall context compression base model is a basic context compression model with the integration of block compression. Therefore, we train a new basic context compression base model without block compression with the same data and pre-training-fine-tuning pipeline with the overall context compression base model. We thereby can analyze the influence of block compression through comparing the performance of these two base models under the same length of the compressed summary token sequence. We follow AutoCompressor (Chevalier et al., 2023) to set the length of summary sequences to 50.

Table 5 demonstrates experiment results. Apart from the plain setting of separately compressing each documentation, we add another setting to observe the performance under a higher compression ratio, where we regard the concatenation of multiple documentation as a whole. With either setting, block compression significantly improves basic context compression.

4.6 Effects of Selective Compression Strategy

To study the effects of selective compression strategy and the necessity of selective strategy in pre-training, we evaluate all four cases of strategy combination during pre-training and fine-tuning. Results are shown in Table 4, which can be seen as an extended version of Table 2.

We find that introducing selective compression benefits the performance even if the final compression manner is overall compression. Moreover, the base model can adapt to selective compression through fine-tuning, though having a performance gap to the best combination. To reach the best performance, the model should use selective compression from the beginning to the end. The conclusion

is supported by Table 4 both with and without the reconstruction loss.

5 Conclusion

In this work, we propose an approach to compress tool documentation into concise and precise summary sequences. There are two main challenges in achieving our gold. First, context compression approaches suffer from uncontrollable compression loss, leading to key information loss. Second, existing approaches cannot generate summary sequences of variable length, and thus cannot support preset compression ratio. We propose two compression strategies to deal with these challenges. To avoid key information loss, we propose the selective compression strategy which allows the key information to be retained as raw text tokens. To support preset compression ratios, we propose block compression which chunks the full sequence to be compressed into small blocks to realize variable length compression upon fixed length compression model.

We evaluate our approach in two tool-using benchmarks. Results show that our method can reach at least comparable performance to the baseline without compression, while achieving up to 16x compression ratio. Furthermore, we explore the effects of our two compression strategies, and provide a training recipe of context compression for tool-using language models as a result.

Limitations

The main limitation of our work is that the proposed strategies rely on human priors. Specifically, the definition of key information and the preset compression ratio need to be tuned according to the actual task.

Another limitation is that our compression approach needs to pre-train the model, which means the computation cost is relatively high, and the approach is not suitable for black box models. Although this is a universal problem of soft context compression approaches, it still hinders the flexibility of our approach.

Acknowledgements

We gratefully acknowledge the support of the National Natural Science Foundation of China (NSFC) via grants 62236004 and 62206078.

References

Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. 2022. Recurrent memory transformer. *Advances in Neural Information Processing Systems*, 35:11079–11091.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023. Binding language models in symbolic languages.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. 2023. Adapting language models to compress contexts. *arXiv preprint arXiv:2305.14788*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

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*.

Together Computer. 2023. Redpajama: An open source recipe to reproduce llama training dataset.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models.

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*.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. 2022. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023a. Llmlingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*.

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

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.

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

Yucheng Li, Bo Dong, Chenghua Lin, and Frank Guerin. 2023b. Compressing context to enhance inference efficiency of large language models. *arXiv preprint arXiv:2310.06201*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2021. Webgpt: Browser-assisted question-answering with human feedback.

OpenAI. 2023. Chatgpt plugins.

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*.

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

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 preprint arXiv:2302.04761*.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. 2023. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama.

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

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`.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, YaGuang Li, Hongrae Lee, Huaixiu Steven Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Vincent Zhao, Yanqi Zhou, Chung-Ching Chang, Igor Krivokon, Will Rusch, Marc Pickett, Pranesh Srinivasan, Laichee Man, Kathleen Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Soraker, Ben Zevenbergen, Vinodkumar Prabhakaran, Mark Diaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravi Rajakumar, Alena Butryna, Matthew Lamm, Viktoriya Kuzmina, Joe Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Aguera-Arcas, Claire Cui, Marian Croak, Ed Chi, and Quoc Le. 2022. Lamda: Language models for dialog applications.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models.

Guan Wang, Sijie Cheng, Xianyuan Zhan, Xiangang Li, Sen Song, and Yang Liu. 2023. Openchat: Advancing open-source language models with mixed-quality data. *arXiv preprint arXiv:2309.11235*.

Yumeng Wang and Zhenyang Xiao. 2024. Loma: Lossless compressed memory attention. *arXiv preprint arXiv:2401.09486*.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. 2023. On the tool manipulation capability of open-source large language models. *arXiv preprint arXiv:2305.16504*.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023. Gpt4tools: Teaching large language model to use tools via self-instruction. *arXiv preprint arXiv:2305.18752*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models.

A Additional Results

A.1 The Necessity of Non-key Information

Although we use open datasets from existing work, there not exists evidence supporting the necessity of non-key information such as the descriptions. Theoretically, the model may guess the functionality of a tool only using the names of the APIs and parameters.

Therefore, we conduct experiments on the case that non-key information is deleted. In other words, the tool documentation consist of almost only the names.

The results are shown in Table **??**. To understand the results, please note that tools in the API-Bank test set are unseen during training. In contrast, APIBench shares the same tool library during training and testing. Thus, when fine-tuned and tested with the same type of input, the model has the chance to memorize the tools in APIBench. However, a performance gap also exists in this case.

Dataset	Fine-tuned Input	Test Input	Accuracy
API-Bank	Full	Full	**71.47**
	Full	Key-only	53.21
	Key-only	Full	59.38
	Key-only	Key-only	40.62
APIBench	Full	Full	**88.24**
	Full	Key-only	67.45
	Key-only	Full	27.51
	Key-only	Key-only	83.75

Table 6: Deleting non-key information significantly hurts the performance, which supports the necessity of non-key information.

To summarize, we can conclude that the non-key information is necessary.

A.2 Comparison with Hard Summarization

Apart from soft compression which we use, another way to achieve tool documentation compression is hard compression, such as prompting ChatGPT to summarize the documentation. However, hard tokens are able to carry less information in the same context length, which is the reason we consider soft compression at first.

To illustrate the manner of hard compression, we use GPT-4 Turbo to compress the non-key part of the tool documentation, which is a very close setting to our main experiment despite we use ChatGPT as the compressor.

We explain the experiment details on API-Bank, which are highly similar to the case on APIBench. We compress all `description` fields since other non-key fields like datatype consist of very short text. Specifically, we use the following prompt and switch on the JSON-only output mode:

```
Here is an API document.


{{json.dumps(the_documentation_dict,
↪  ensure_ascii=False)}}


Replace each "description" field with a
↪  brief summary and keep the other
↪  parts as is. The summary should
↪  remove redundancy and express the
↪  text as concisely as possible,
↪  ensuring that allkey information are
↪  preserved. Only output a single json
↪  without the quote block.
```

Also, we carefully check the output and leverage regenerating to ensure any other field is kept as is.

To provide better intuition, we give an example as follows (formatted for easy reading).

```
# the raw doc
{
    "name": "Translate",
    "description": "Translate the text
    ↪  to the target language.",
    "input_parameters": {
        "src": {
            "type": "str",
            "description": "The text to
            ↪  be translated."
        },
        "src_lang": {
            "type": "str",
            "description": "[Optional]
            ↪  The source language to
            ↪  translate from. Default
            ↪  is auto."
        },
        "tgt_lang": {
            "type": "str",
            "description": "[Optional]
            ↪  The target language to
            ↪  translate to. Default is
            ↪  english/en."
        }
    },
    "output_parameters": {
        "translated_text": {
            "type": "str",
            "description": "The
            ↪  translated text."
        }
    }
}

# doc after GPT-4 summarization
{
    "name": "Translate",
    "description": "Translates text to a
    ↪  specified language.",
    "input_parameters": {
        "src": {
            "type": "str",
            "description": "Text for
            ↪  translation."
        },
        "src_lang": {
            "type": "str",
```

```
            "description": "Source
            ↪  language (auto by
            ↪  default)."
        },
        "tgt_lang": {
            "type": "str",
            "description": "Target
            ↪  language (English by
            ↪  default)."
        }
    },
    "output_parameters": {
        "translated_text": {
            "type": "str",
            "description": "Resulting
            ↪  translation."
        }
    }
}
```

We can see from the example that plain text cannot compress concise input with a high compression ratio, while soft compression has the chance to compress a short description into a single soft token at an over 10× compression ratio.

The only difference on APIBench is that we compress the `example_code` as well as the `description` field. The other fields are always very short.

Next, we fine-tune using the baseline setting in the paper, namely fine-tuning a LLaMA-7b model fine-tuned on ShareGPT. To avoid inconsistency between training and testing, we also evaluate the case where the training data are also compressed.

Dataset	Fine-tuned Input	Test Input	Accuracy
API-Bank	Raw	Raw	**71.47**
	Raw	GPT-4 sum	67.87
	GPT-4 sum	Raw	55.53
	GPT-4 sum	GPT-4 sum	52.70
APIBench	Raw	Raw	**88.24**
	Raw	GPT-4 sum	87.79
	GPT-4 sum	Raw	86.27
	GPT-4 sum	GPT-4 sum	86.44

Table 7: GPT-4 Turbo hard summarization perform worse in accuracy than our approach.

Table 7 lists the accuracy and Table 8 lists the averaged compression ratio over the datasets, where we find GPT-4 summarization is less efficient. Tested with GPT-4 summarized input, the baseline model achieves lower performance. When trained with GPT-4 summarized data, the performance goes even lower. Note that the official training set of API-Bank uses a dedicated set of tool documentation which are model generated thus far more messy and harder to summarize than the test data. In contrast, APIBench uses the same set of tool documentation in the training set and the test set. These results imply a possible flaw that untrained hard summarization may introduce information loss or error on tool-using tasks.

Approach	Dataset	Achieved Compression Ratio
GPT-4 sum	API-Bank	1.39×
	APIBench	1.42×
Ours	(nearly) dataset agnostic	configurable, up to 16× at least

Table 8: GPT-4 Turbo hard summarization achieves lower compression ratios than our approach.

Please note that our work mainly focuses on developing an efficient approach to compressing tool documentation based on the soft context compression framework, instead of a comparison of soft/hard compression approaches on specific tasks. Therefore, the results of hard summary baselines act as additional data to help readers better understand our motivation and better illustrate the differences between soft/hard compression approaches. Anyway, the performance of hard summarization/compression has a limited relation with the integrity of our work.