Generative Adapter: Contextualizing Language Models in Parameters with A Single Forward Pass

Tong Chen[♣*] **Hao Fang**[♡] **Patrick Xia**[♡] **Xiaodong Liu**[♠] **Benjamin Van Durme**[♡]
Luke Zettlemoyer[♣] **Jianfeng Gao**[♠] **Hao Cheng**[♠†]
[♣] University of Washington [♡] Microsoft [♠] Microsoft Research

Abstract

Large language models (LMs) acquire substantial knowledge during pretraining but often need adaptation to new contexts, tasks, or domains, typically achieved through fine-tuning or prompting. However, fine-tuning incurs significant training costs, while prompting increases inference overhead. Inspired by fast weight memory, we introduce `GenerativeAdapter`, an effective and efficient adaptation method that encode test-time context into LM's parameters with a single forward pass. `GenerativeAdapter` augments a frozen pretrained LM with a lightweight adapter generator, trained via self-supervised learning, to produce parameter-efficient adapters. Notably, our generator is general-purpose, *i.e.,* one generator can adapt the corresponding base model for all langauge processing scenarios. We apply `GenerativeAdapter` to two pretrained LMs (Mistral-7B and Llama2-7B) and evaluate the adapted models across knowledge acquisition from documents, learning from demonstrations, and personalization for users. Overall, `GenerativeAdapter` provides a viable solution for adapting large LMs to evolving information and providing tailored user experience, while reducing training and inference costs relative to traditional fine-tuning and prompting techniques.

1 Introduction

Adaptation is essential for language models (LMs) to acquire new world knowledge [16, 14, 26], learn new tasks [27], and personalize to individual users [34]. The predominant adaptation methods typically involve either *prompting* or *fine-tuning* [4]. As the scale of LMs continues to increase, adapting them becomes increasingly difficult due to efficiency constraints during both training and inference times [13, 5, 43, 1]. Thus, we are interested in exploring alternative approaches for effectively and efficiently adapting pretrained LMs.

In this work, we present `GenerativeAdapter`, a novel method for training a neural network (adapter generator) to generate adapters that contextualize pretrained LMs on-the-fly with temporary knowledge from incoming contexts. Inspired by fast weights [2, 36, *inter alia*], our approach incorporates a lightweight adapter generator on top of pretrained LM as the slow network to produce updated parameters for the fast network (the adapted LM). As far as we know, we are the first to explore this direction. Specifically, the pretrained base LM remains frozen while we train the LM-specific adapter generator to generate layer-by-layer additive updates, similar to recent parameter-efficient fine-tuning (PEFT) techniques [12, 13]. For each layer, a bilinear adapter generator network uses the outer product of past context hidden states from the corresponding base LM layer to generate delta weights. These generated delta weights are then added to the base LM weights to form an adapted LM for future predictions. Similar to previous work on fast weights, our method achieves test-time

[*]Work done during internship at Microsoft Research.

[†]Correspondence to {chentong@cs.washington.edu, chehao@microsoft.com}

NeurIPS 2024 Workshop on Adaptive Foundation Models (NeurIPS 2024).

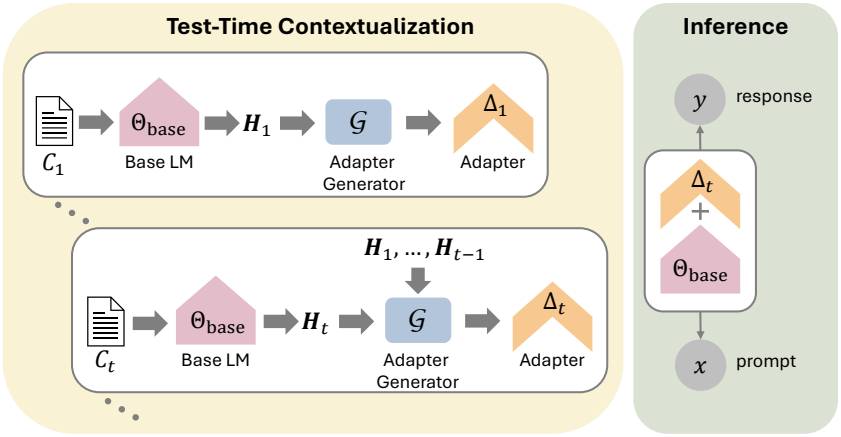


Figure 1: Overview of `GenerativeAdapter`. **Left**: During test-time contextualization, the adapters $\Delta_1, \ldots, \Delta_t$ are generated sequentially for the stream of context chunks $C_1, \ldots, C_t$. At a given time step t, the context chunk C_t is encoded by the base LM Θ_{base} into hidden state vectors $\mathbf{H}_t$. Then the generator $\mathcal{G}$ produces a new adapter Δ_t based on the collection of hidden state vectors $\mathbf{H}_1, \ldots, \mathbf{H}_t$ representing the accumulated context. **Right**: During inference, we combine the latest adapter Δ_t with the base LM Θ_{base} to generate responses for input prompts.

adaptation using only forward passes, allowing dynamic updates as new context arrives in sequential chunks. We train the generator end-to-end in a self-supervised manner by compressing the context into a generated adapter and then computing the next-token prediction loss on a target sequence using the adapted LM. Once trained, our method can produce adapted LMs that effectively capture knowledge from the context to solve multiple downstream tasks, thus improving the adaptability of off-the-shelf pretrained LMs.

We evaluate our method on two scenarios where on-the-fly contextualizing pretrained LMs is crucial: acquiring new factual knowledge and learning from demonstrations. These scenarios involve diverse forms of context with varying lengths, including documents with background knowledge, task-specific input-output examples and user-specific conversations. For knowledge acquisition, `GenerativeAdapter` effectively memorizes factual knowledge from provided documents, with minimal loss compared to full-context prompting at short context lengths. Notably, our method excels in memorizing long-context documents, managing to handle context lengths up to 32K on StreamingQA [24] and 8K on SQuAD [31] better than continual pretraining. On MetaICL, `GenerativeAdapter` follows demonstrations effectively, achieving superior accuracy compared to its base model's in-context learning, underscoring the model's ability to adapt to new tasks efficiently. For cases with many queries from the same user on edge devices, the benefits of our method are more evident, positioning our method as a viable tool for personalized LMs.

2 Method

We present `GenerativeAdapter`, an efficient and effective framework for directly generating additive weight updates to contextualize the pretrained LM (a *frozen* base LM) at test time. Unlike continual pretraining and supervised fine-tuning which update the pretrained LM via gradient descent, our method achieves adaptation using one forward passes only.

Adaptation with Test-time Contextualization To *contextualize* a base model, Θ_{base}, to a given context C, our goal is to obtain an updated model, Θ_C, that can respond to user instructions using the information provided in the context C. The context can include different types of data, such as documents, dialogues, or task-specific few-shot demonstrations. We specifically focus on *test-time contextualization*, where context arrives incrementally as a stream of data, such as a continuous flow of documents or dialogue sessions. We represent this streaming context up to time step t as $\Sigma(t) := (C_1, \ldots, C_t)$, where C_t is the context chunk arriving at time step t. In this online adaptation scenario, the model must be efficiently adapted to each new context chunk as it becomes available.

Generative Adapter As shown in Figure 1, we propose `GenerativeAdapter`, a framework that adapts the base model Θ_{base} to new contexts through a single forward pass for each context chunk.

Given test-time context $\Sigma(t)$, the base model is adapted to $\Theta_{\Sigma(t)} = \Theta_{\text{base}} + \Delta_t$, where Δ_t is a context-dependent additive adapter. The adapted model $\Theta_{\Sigma(t)}$ is then used for inference on inputs relevant to $\Sigma(t)$, such as summarizing or answering questions about a user's past conversations. We propose a learned adapter generator $\mathcal{G}$ to produce Δ based on the streaming context Σ. The generator converts context, encoded by the base LM, into the parameter matrices (*e.g.,* key/query/value/output layers of the multi-head attention and down/up projection layers of the feed-forward network).

A linear projection layer in the l-th Transformer block ($l = 1, 2, \ldots, L$) is $\mathbf{o} = \mathbf{W}^{(l)}\mathbf{h}$, where $\mathbf{W}^{(l)} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ is the weight matrix, $\mathbf{h} \in \mathbb{R}^{d_{\text{in}}}$ is the input vector, and $\mathbf{o} \in \mathbb{R}^{d_{\text{out}}}$ is the output vector. We omit the bias term for simplicity. For the adapted LM, $\mathbf{W}^{(l)} = \mathbf{W}^{(l)}_{\text{base}} + \mathbf{W}^{(l)}_{\Delta}$, where $\mathbf{W}^{(l)}_{\text{base}}$ and $\mathbf{W}^{(l)}_{\Delta}$ are from the base model Θ_{base} and the context-dependent adapter Δ, respectively. The weights of Δ are generated using the streaming context Σ encoded by Θ_{base}, resulting in hidden states $\mathbf{h}^{(l)}_1, \mathbf{h}^{(l)}_2, \ldots, \mathbf{h}^{(l)}_M \in \mathbb{R}^{d_h}$, packed in a matrix $\mathbf{H}^{(l)} \in \mathbb{R}^{M \times d_h}$. The weights $\mathbf{W}^{(l)}_{\Delta}$ are then generated as $\mathbf{W}^{(l)}_{\Delta} = \mathcal{G}^{(l)}(\mathbf{H}^{(l-1)})$, where $\mathcal{G}^{(l)}(\cdot)\colon \mathbb{R}^{*\times d_h} \to \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ transforms any hidden state sequence into a fixed-dimensional weight matrix. We omit the layer index l when unambiguous. To avoid dependency on M, we use a bi-linear function:

$$\mathbf{W}_{\Delta} = \mathcal{G}(\mathbf{H}) = (\mathbf{A}_1\mathbf{A}_2)\mathbf{H}^{\top}\mathbf{H}(\mathbf{B}_1\mathbf{B}_2) = (\mathbf{A}_1\mathbf{A}_2)(\sum_{m=1}^{M} \mathbf{h}_m \otimes \mathbf{h}_m)(\mathbf{B}_1\mathbf{B}_2), \tag{1}$$

where $\otimes$ is the outer product, and $\mathbf{A}_1 \in \mathbb{R}^{d_{\text{out}} \times d_r}$, $\mathbf{A}_2 \in \mathbb{R}^{d_r \times d_h}$, $\mathbf{B}_1 \in \mathbb{R}^{d_h \times d_r}$, $\mathbf{B}_2 \in \mathbb{R}^{d_r \times d_{\text{in}}}$ are learnable parameters, with $d_r \ll d_{\text{in}}, d_{\text{out}}, d_h$ to limit the parameter count.

Learning to Update with Self-supervised Pretraining To preserve the language modeling capability of the adapted models $\Theta_{\Sigma(t)}$ for $t \in \{1, 2, \ldots\}$, we pretrain the weight generator $\mathcal{G}$ using the next-token prediction loss of $\Theta_{\Sigma(t)}$ in a self-supervised manner on web corpora. In other words, the adapter generator is trained on top of the frozen base model Θ_{base} in an end-to-end fashion. Specifically, we use two self-supervision pretraining tasks: *reconstruction* and *completion*.

The reconstruction task [9, 30] draws inspiration from autoencoders and aims to train the weight generator $\mathcal{G}$ to embed contextual information into the generated weights. This process compresses the input context $(x_1, \ldots, x_m)$ into a generated adapter, $\mathcal{G}(x_{1:m})$, which is subsequently used to reconstruct the input. Formally, this is accomplished by maximizing the log-likelihood of the input tokens with the adapted LM, using weights updated from the same text: $\mathcal{L}_{\text{reconstruction}}(\mathcal{G}) = \log P\left(x_1, \ldots, x_m \mid \Theta_{\text{base}} + \mathcal{G}(x_{1:m})\right)$. The completion task [44, 19] trains the adapted LM to generate the continuation of the given context. The goal is to maximize the log-likelihood of tokens $x_{m+1}, \ldots, x_n$, which represent the continuation of the context $x_1, \ldots, x_m$ in the dataset: $\mathcal{L}_{\text{completion}}(\mathcal{G}) = \log P\left(x_{m+1}, \ldots, x_n \mid \Theta_{\text{base}} + \mathcal{G}(x_{1:m})\right)$. We observe using both of the task can make the generated adapter memorize and utilize the contextual information. Similar to prior work [9], the generator is trained to maximize the sum of the objective functions of the two task: $\max_{\mathcal{G}} \mathcal{L}_{\text{reconstruction}}(\mathcal{G}) + \mathcal{L}_{\text{completion}}(\mathcal{G})$.

Dynamic Streaming Updating In practice, the context can arrive in chunks sequentially. The matrix of hidden states H_t at step t is computed based on all previous context chunks $\Sigma(t-1)$. This hidden state is then used to generate an adapter for the current chunk $\Sigma(t)$, which, in turn, is also used to compute the hidden states for future context steps. Based on Equation 1, to compute the adapter of $\Sigma(t)$ we need to concatenate all hidden states (*i.e.,* $[\mathbf{H}_1; \ldots; \mathbf{H}_t] \in \mathbb{R}^{(M_1 + \cdots + M_t) \times d_h}$) of the context chunks in $\Sigma(t)$ to generate the adapter, *i.e.,* $\mathbf{W}_{\Delta_t} = \mathcal{G}([\mathbf{H}_1; \ldots; \mathbf{H}_t])$. Fortunately, our formulation allows an efficient updating mechanism without explicitly storing history hidden states, noting that

$$\mathbf{W}_{\Delta_t} = (\mathbf{A}_1\mathbf{A}_2)([\mathbf{H}_1; \ldots; \mathbf{H}_t]^{\top}[\mathbf{H}_1; \ldots; \mathbf{H}_t])(\mathbf{B}_1\mathbf{B}_2) = (\mathbf{A}_1\mathbf{A}_2)\left(\sum_{i=1}^{t} \mathbf{H}_i^{\top}\mathbf{H}_i\right)(\mathbf{B}_1\mathbf{B}_2). \tag{2}$$

Thus, the update can be efficiently computed. See Appendix for details.

SVD Normalization In preliminary experiments, we find that using the naive outer product for generating weights led to instability during training, causing convergence issues. When multiplying the generated matrix with the input vector, the resulting output can either diminish to near-zero or grow excessively large.

To address this instability, we introduce normalization into the formulation, $\mathbf{W}_{\Delta_t} = \mathbf{A}_1 \operatorname{norm}\left(\mathbf{A}_2 \left(\sum_{i=1}^{t} \mathbf{H}_i^\top \mathbf{H}_i\right) \mathbf{B}_1\right) \mathbf{B}_2$.

Our pilot experiments find that normalization based on singular value decomposition (SVD) is particular effective, among other normalization strategies. The SVD normalization technique ensures the singular values of the outer product are normalized to 1. Given a matrix $\mathbf{M}$, we define SVD normalization as: $\operatorname{norm}(\mathbf{M}) = \mathbf{U}\mathbf{V}^\top$, where $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ is the SVD factorization. This normalization resets the positive singular values of the matrix to one, preventing the vectors from excessively shrinking or exploding.

An additional benefit of SVD normalization is that it can naturally produce low-rank matrices. Instead of performing a full-rank decomposition, we approximate the input matrix with a rank-r SVD decomposition, where r is a hyperparameter set in advance. Consequently, the matrix can be written as the product of two low-rank matrices, similar to a LoRA adapter [13]: $\mathbf{W}_{\Delta_t} = \mathbf{A}_1 \operatorname{norm}(\mathbf{S}_t)\mathbf{B}_2 = (\mathbf{A}_1 U(\mathbf{H}))(V^\top(\mathbf{H})\mathbf{B}_2)$, where $U(\mathbf{H})$ and $V(\mathbf{H})$ are the matrices resulting from SVD normalization. This low-rank approximation reduces both computational cost and memory usage.

3 Experiments

We experiment with using both Mistral-7B-Instruct (v0.2) [15] and Llama2-7B-Chat [40] as the base LMs. We pretrain the model on 1B tokens from SlimPajama [38] and instruction fine-tune on a mixture of tasks including question answering, summarization, etc. We evaluate `GenerativeAdapter` on two representative scenarios: acquiring new factual knowledge from documents and learning from demonstrations. We show more details about training and results on personalization and ablation study in Appendix.

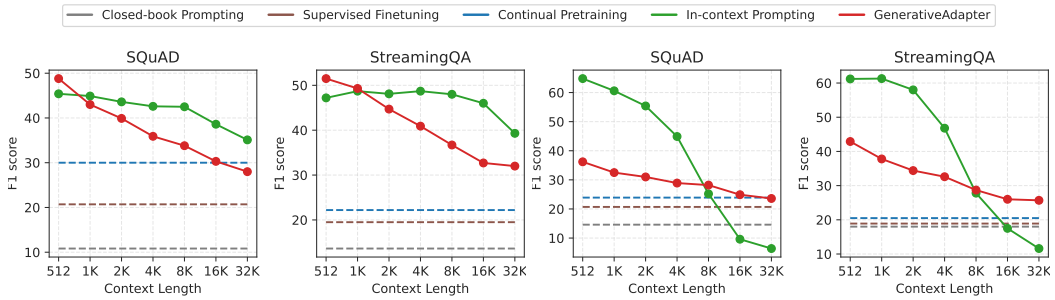
Document-based Question Answering with Varying Context Length. We evaluate the fact memorization ability of the adapted model on two question answering (QA) datasets, SQuAD [31] and StreamingQA [24]. We split the passages into groups with each group having an average length of k tokens ($k \in \{512, 1K, 2K, 4K, 8K, 16K, 32K\}$). The base model is adapted on each group, and the adapted model is evaluate on questions associated with the group. F1 score between the output and gold answer is used as the metric for evaluation [31].

For baseline, we consider both full parameter fine-tuning and full context prompting using the same base model. We use both full parameter fine-tuning and full context prompting on the same base model as baselines. For fine-tuning, we consider two variants: *supervised fine-tuning* (SFT) trains the model on a training set of question-answer pairs from articles distinct from the test set. Continual pretraining (CPT) first trains on all documents in the test set, followed by SFT on the training set. Inference is conducted in a closed-book manner, *i.e.,* the model directly generates answers. For prompting, if the context exceeds Llama2-7B-Chat's 4K token limit, we truncate to the last 4K tokens.

We present QA accuracy results for SQuAD and StreamingQA and the computation costs for StreamingQA in Figure 2 and Figure 4, respectively. Fine-tuning methods (SFT and CPT) show constant QA performance in a closed-book setting, while `GenerativeAdapter` and prompting are evaluated with varying context lengths, where recalling facts becomes harder with longer contexts. `GenerativeAdapter` achieves improved QA accuracy, particularly with short contexts ($<$ 1K tokens), and avoids the additional inference cost of prompting, which grows with context length due to attention (shown by green lines in Figure 4). `GenerativeAdapter` outperforms CPT for contexts below 8K tokens and requires significantly less preprocessing time (Figure 4).

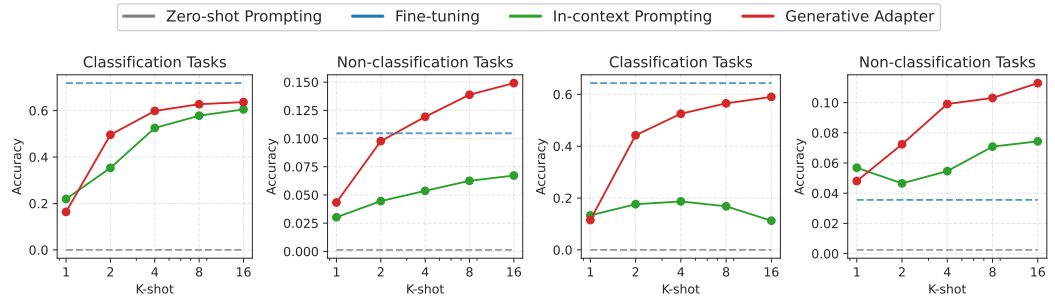
In-Context Learning with varying in-context examples We evaluate the in-context learning ability of `GenerativeAdapter` on MetaICL [27], consisting of 26 test tasks. None of these test tasks were seen during the training of adapter generator. For each task, we use 1, 2, 4, 8, and 16 demonstrations randomly sampled from the corresponding dev split following the MetaICL evaluate pipeline. `GenerativeAdapter` encode the concatenation of the demonstrations into a generator.

We compare three baselines: 1) zero-shot prompting with the base LM using only task instructions; 2) few-shot prompting with the base LM [27], where each test case is prepended with a few-shot example; and 3) fine-tuning the base LM on each evaluation task using 16 input-output pairs, equivalent to the maximum number of shots in our evaluation.



(a) Mistral-7B-Instruct (b) Llama2-7B-Chat

Figure 2: Document QA Performance on SQuAD and StreamingQA across varying context lengths. For each point, the QA accuracy (F1 score) is calculated based on the same set of test questions. Both fine-tuning methods (supervised fine-tuning and continuous pretraining) are evaluated in a closed-book manner with constant QA performance across varying context lengths.



(a) Mistral-7B-Instruct (b) Llama2-7B-Chat

Figure 3: Accuracy plots on MetaICL with varying K-shot in-context examples. Both fine-tuned and zero-shot prompting baselines are instructed to complete the task without any in-context examples.

Figure 3 summarizes MetaICL results for classification and non-classification tasks with varying numbers of in-context examples. Fine-tuned models (blue) and zero-shot baselines (grey) perform consistently across shots without examples. Fine-tuning yields higher accuracy in classification but struggles with non-classification, suggesting 16 shots are insufficient for learning the output style. `GenerativeAdapter` consistently outperforms few-shot prompting, particularly on challenging non-classification , indicating our method enhances the base model's in-context learning.

4 Conclusion

In this work, we introduce `GenerativeAdapter`, a method for efficiently adapting pretrained LMs on-the-fly using test-time context through forward passes only. We design a bilinear adapter generator network on top of frozen pretrained LMs, transforming text contexts into updated model parameters. Our adapter generator network is trained end-to-end with the frozen pretrained LM using two self-supervised tasks on web corpora. We assess `GenerativeAdapter` across three scenarios that benefit from contextualizing pretrained LMs: acquiring new factual knowledge, learning from demonstrations, and personalizing to individual users. Our experiments indicate that `GenerativeAdapter` reduces information loss compared to full-context prompting in retaining factual knowledge from context documents. Additionally, the model effectively adapts to new task instructions when learning from demonstrations. Finally, `GenerativeAdapter` achieves comparable fact recall performance to efficient prompting methods while utilizing lower inference-time computation, showcasing its feasibility for user-specific adaptation in personalization scenarios. For future work, it would be interesting to further explore scaling up the adapter generator, such as by integrating adapters into additional layers, and to investigate more selective update rules [35].

References

[1] Z. Allen-Zhu and Y. Li. Physics of language models: Part 3.1, knowledge storage and extraction, 2024.

[2] J. Ba, G. E. Hinton, V. Mnih, J. Z. Leibo, and C. Ionescu. Using fast weights to attend to the recent past. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[3] P. Bajaj, D. Campos, N. Craswell, L. Deng, J. Gao, X. Liu, R. Majumder, A. McNamara, B. Mitra, T. Nguyen, M. Rosenberg, X. Song, A. Stoica, S. Tiwary, and T. Wang. Ms marco: A human generated machine reading comprehension dataset, 2018.

[4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[5] A. Chevalier, A. Wettig, A. Ajith, and D. Chen. Adapting language models to compress contexts. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3829–3846, Singapore, Dec. 2023. Association for Computational Linguistics.

[6] K. M. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Q. Davis, A. Mohiuddin, L. Kaiser, D. B. Belanger, L. J. Colwell, and A. Weller. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021.

[7] K. Clark, K. Guu, M.-W. Chang, P. Pasupat, G. Hinton, and M. Norouzi. Meta-learning fast weight language models. In Y. Goldberg, Z. Kozareva, and Y. Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9751–9757, Abu Dhabi, United Arab Emirates, Dec. 2022. Association for Computational Linguistics.

[8] D. Dua, Y. Wang, P. Dasigi, G. Stanovsky, S. Singh, and M. Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2368–2378, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[9] T. Ge, H. Jing, L. Wang, X. Wang, S.-Q. Chen, and F. Wei. In-context autoencoder for context compression in a large language model. In *The Twelfth International Conference on Learning Representations*, 2024.

[10] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey, 2024.

[11] G. E. Hinton and D. C. Plaut. Using fast weights to deblur old memories. In *Proceedings of the ninth annual conference of the Cognitive Science Society*, pages 177–186, 1987.

[12] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. de Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-efficient transfer learning for nlp. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[13] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[14] N. Hu, E. Mitchell, C. Manning, and C. Finn. Meta-learning online adaptation of language models. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4418–4432, Singapore, Dec. 2023. Association for Computational Linguistics.

[15] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed. Mistral 7b, 2023.

[16] Z. Jiang, Z. Sun, W. Shi, P. Rodriguez, C. Zhou, G. Neubig, X. Lin, W.-t. Yih, and S. Iyer. Instruction-tuned language models are better knowledge learners. In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5421–5434, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics.

[17] Q. Jin, B. Dhingra, Z. Liu, W. Cohen, and X. Lu. PubMedQA: A dataset for biomedical research question answering. In K. Inui, J. Jiang, V. Ng, and X. Wan, editors, *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 2567–2577, Hong Kong, China, Nov. 2019. Association for Computational Linguistics.

[18] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are RNNs: Fast autoregressive Transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[19] J.-H. Kim, J. Yeom, S. Yun, and H. O. Song. Compressed context memory for online language model interaction. In *The Twelfth International Conference on Learning Representations*, 2024.

[20] T. Kočiský, J. Schwarz, P. Blunsom, C. Dyer, K. M. Hermann, G. Melis, and E. Grefenstette. The NarrativeQA reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018.

[21] W. Kryscinski, N. Rajani, D. Agarwal, C. Xiong, and D. Radev. BOOKSUM: A collection of datasets for long-form narrative summarization. In Y. Goldberg, Z. Kozareva, and Y. Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 6536–6558, Abu Dhabi, United Arab Emirates, Dec. 2022. Association for Computational Linguistics.

[22] X. L. Li and P. Liang. Prefix-tuning: Optimizing continuous prompts for generation. In C. Zong, F. Xia, W. Li, and R. Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online, Aug. 2021. Association for Computational Linguistics.

[23] X. V. Lin, X. Chen, M. Chen, W. Shi, M. Lomeli, R. James, P. Rodriguez, J. Kahn, G. Szilvasy, M. Lewis, L. Zettlemoyer, and W. tau Yih. RA-DIT: Retrieval-augmented dual instruction tuning. In *The Twelfth International Conference on Learning Representations*, 2024.

[24] A. Liska, T. Kocisky, E. Gribovskaya, T. Terzi, E. Sezener, D. Agrawal, C. De Masson D'Autume, T. Scholtes, M. Zaheer, S. Young, E. Gilsenan-Mcmahon, S. Austin, P. Blunsom, and A. Lazaridou. StreamingQA: A benchmark for adaptation to new knowledge over time in question answering models. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 13604–13622. PMLR, 17–23 Jul 2022.

[25] S.-Y. Liu, C.-Y. Wang, H. Yin, P. Molchanov, Y.-C. F. Wang, K.-T. Cheng, and M.-H. Chen. DoRA: Weight-decomposed low-rank adaptation. In *Proceedings of the 41th International Conference on Machine Learning*, 2024.

[26] N. Mecklenburg, Y. Lin, X. Li, D. Holstein, L. Nunes, S. Malvar, B. Silva, R. Chandra, V. Aski, P. K. R. Yannam, T. Aktas, and T. Hendry. Injecting new knowledge into large language models via supervised fine-tuning, 2024.

[27] S. Min, M. Lewis, L. Zettlemoyer, and H. Hajishirzi. MetaICL: Learning to learn in context. In M. Carpuat, M.-C. de Marneffe, and I. V. Meza Ruiz, editors, *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2791–2809, Seattle, United States, July 2022. Association for Computational Linguistics.

[28] C. Packer, S. Wooders, K. Lin, V. Fang, S. G. Patil, I. Stoica, and J. E. Gonzalez. Memgpt: Towards llms as operating systems, 2024.

[29] H. Peng, N. Pappas, D. Yogatama, R. Schwartz, N. Smith, and L. Kong. Random feature attention. In *International Conference on Learning Representations*, 2021.

[30] G. Qin, C. Rosset, E. Chau, N. Rao, and B. Van Durme. Dodo: Dynamic contextual compression for decoder-only LMs. In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9961–9975, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics.

[31] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. SQuAD: 100,000+ questions for machine comprehension of text. In J. Su, K. Duh, and X. Carreras, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, Nov. 2016. Association for Computational Linguistics.

[32] S. Reddy, D. Chen, and C. D. Manning. CoQA: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266, 2019.

[33] A. Rogers, O. Kovaleva, M. Downey, and A. Rumshisky. Getting closer to AI complete question answering: A set of prerequisite real tasks. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8722–8731. AAAI Press, 2020.

[34] A. Salemi, S. Mysore, M. Bendersky, and H. Zamani. LaMP: When large language models meet personalization. In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7370–7392, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics.

[35] I. Schlag, K. Irie, and J. Schmidhuber. Linear Transformers are secretly fast weight programmers. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[36] J. Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

[37] J. Schmidhuber. Reducing the ratio between learning complexity and number of time varying variables in fully recurrent nets. In *Proceedings of International Conference on Artificial Neural Networks (ICANN)*, pages 460–463, 1993.

[38] D. Soboleva, F. Al-Khateeb, R. Myers, J. R. Steeves, J. Hestness, and N. Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama, 2023.

[39] J. Tack, J. Kim, E. Mitchell, J. Shin, Y. W. Teh, and J. R. Schwarz. Online adaptation of language models with a memory of amortized contexts, 2024.

[40] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[41] Y.-H. H. Tsai, S. Bai, M. Yamada, L.-P. Morency, and R. Salakhutdinov. Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In K. Inui, J. Jiang, V. Ng, and X. Wan, editors, *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 4344–4353, Hong Kong, China, Nov. 2019. Association for Computational Linguistics.

[42] J. Xu, A. Szlam, and J. Weston. Beyond goldfish memory: Long-term open-domain conversation. In S. Muresan, P. Nakov, and A. Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5180–5197, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[43] Z. Yang, N. Band, S. Li, E. Candès, and T. Hashimoto. Synthetic continued pretraining, 2024.

[44] P. Zhang, Z. Liu, S. Xiao, N. Shao, Q. Ye, and Z. Dou. Compressing lengthy context with ultragist, 2024.

[45] Q. Zhang, M. Chen, A. Bukharin, P. He, Y. Cheng, W. Chen, and T. Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*, 2023.

A Method Details

A.1 Dynamic Streaming Update

In practice, the context can arrive in chunks sequentially. The matrix of hidden states H_t at step t is computed based on all previous context chunks $\Sigma(t-1)$. This hidden state is then used to generate an adapter for the current chunk $\Sigma(t)$, which, in turn, is also used to compute the hidden states for future context steps. Based on Equation 1, to compute the adapter of $\Sigma(t)$ we need to concatenate all hidden states (*i.e.,* $[\mathbf{H}_1; \ldots; \mathbf{H}_t] \in \mathbb{R}^{(M_1+\cdots+M_t)\times d_h}$) of the context chunks in $\Sigma(t)$ to generate the adapter, *i.e.,* $\mathbf{W}_{\Delta_t} = \mathcal{G}([\mathbf{H}_1; \ldots; \mathbf{H}_t])$. Fortunately, our formulation allows an efficient updating mechanism without explicitly storing history hidden states, noting that

$$\mathbf{W}_{\Delta_t} = (\mathbf{A}_1\mathbf{A}_2)([\mathbf{H}_1; \ldots; \mathbf{H}_t]^\top [\mathbf{H}_1; \ldots; \mathbf{H}_t])(\mathbf{B}_1\mathbf{B}_2) = (\mathbf{A}_1\mathbf{A}_2)(\sum_{i=1}^{t} \mathbf{H}_i^\top \mathbf{H}_i)(\mathbf{B}_1\mathbf{B}_2). \quad (3)$$

Thus, the update can be efficiently computed as

$$\mathbf{S}_t \leftarrow \mathbf{S}_{t-1} + \mathbf{A}_2\mathbf{H}_t^\top \mathbf{H}_t\mathbf{B}_1 \quad (4)$$

$$\mathbf{W}_{\Delta_t} \leftarrow \mathbf{A}_1\mathbf{S}_t\mathbf{B}_2 \quad (5)$$

where $\mathbf{H}_t \in \mathbb{R}^{M_t\times d_h}$ stores the hidden states for the t-th context chunk, and the partial sum $\mathbf{S}_t \in \mathbb{R}^{d_r\times d_r}$ acts as the memory of history context chunks with $\mathbf{S}_0$ initialized as all zeros. Note directly storing $\mathbf{W}_{\Delta_t} \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$ or $\sum_i \mathbf{H}_i^\top \mathbf{H}_i \in \mathbb{R}^{d_h\times d_h}$ would require much more memory because we control $d_r \ll \min\{d_{\text{in}}, d_{\text{out}}, d_h\}$.

Our preliminary experiments find that this architecture exhibits some empirical instability because the generated matrix W_{Δ_t} can transform an input vector $\mathbf{x}$ into a vector containing values with either extremely large or near-zero magnitudes, due to its skewed distribution of its singular values. In §A.2, we will explain how normalization can address the instability issue.

A.2 Normalization for Generated Weights

In preliminary experiments, we find that using the naive outer product for generating weights led to instability during training, causing convergence issues. When multiplying the generated matrix with the input vector, the resulting output can either diminish to near-zero or grow excessively large.

To address this instability, we introduce normalization into the formulation, *i.e.,*

$$\mathbf{W}_{\Delta_t} \leftarrow \mathbf{A}_1 \,\text{norm}(\mathbf{S}_t)\mathbf{B}_2 = \mathbf{A}_1 \,\text{norm}\left(\mathbf{A}_2 \sum_{i=1}^{t} \left(\mathbf{H}_i^\top \mathbf{H}_i\right) \mathbf{B}_1\right) \mathbf{B}_2. \quad (6)$$

Our pilot experiments find that normalization based on singular value decomposition (SVD) is particular effective, among other normalization strategies.

SVD Normalization The SVD normalization technique ensures the singular values of the outer product are normalized to 1. Given a matrix $\mathbf{M}$, we define SVD normalization as:

$$\text{norm}(\mathbf{M}) = \mathbf{U}\mathbf{V}^\top, \quad (7)$$

where $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ is the SVD factorization. This normalization resets the positive singular values of the matrix to one, preventing the vectors from excessively shrinking or exploding.

Low-Rank SVD and LoRA An additional benefit of SVD normalization is that it can naturally produce low-rank matrices. Instead of performing a full-rank decomposition, we approximate the input matrix with a rank-r SVD decomposition, where r is a hyperparameter set in advance. Consequently, the matrix can be written as the product of two low-rank matrices, similar to a LoRA adapter [13]:

$$\mathbf{W}_{\Delta_t} = \mathbf{A}_1 \,\text{norm}(\mathbf{S}_t)\mathbf{B}_2 \quad (8)$$

$$= (\mathbf{A}_1 U(\mathbf{H}))(V^\top(\mathbf{H})\mathbf{B}_2), \quad (9)$$

where $U(\mathbf{H})$ and $V(\mathbf{H})$ are the matrices resulting from SVD normalization. This low-rank approximation reduces both computational cost and memory usage.

B Dataset Details

Pretraining. We pretrain our models using a randomly sampled subset of 1B tokens from the SlimPajama corpus. For validation, we sample an additional 100 segments, each containing 2K tokens, from the same corpus.

Instruction Tuning. We perform instruction tuning using a combination of question answering, in-context learning, and instruction following datasets, following prior studies [23, 9, 44].

C Training Setup

We experiment with using both Mistral-7B-Instruct (v0.2) [15] and Llama2-7B-Chat [40] as the base LMs. For efficiency, our main experiments train adapter generators to only update the output projection layers of the multi-head attention unit in Transformer. We study a more capable implementation in §F and defer the full exploration of other modules for future work.

Hyperparameters For efficiency, our main experiments train adapter generators to only update the output projection layers of the multi-head attention unit in Transformer. The intermediate dimension d_r and SVD rank r are set to 1,024 and 128, respectively. Approximately, this leads to 500 million parameters for the generator, with the generated adapter of 32 million parameters.

Pipeline Following the standard training pipeline of LM development [15, 40], the training of our adapter generator includes a pretraining phase described in §2 followed by instruction tuning. For pretraining, we randomly sample 1 billion tokens from SlimPajama [38] which are split into segments of 8,192 tokens each. For instruction tuning, we use a mix of tasks such as question answering, in-context learning, and general instruction following, which we ensure that there is no overlap with downstream tasks, with a detailed list provided in the appendix. The context is divided into chunks of 1,024 tokens to utilize the dynamic updating mechanism described in §2.

Implementation We empirically found that normalization is crucial for `GenerativeAdapter` to function effectively. For SVD normalization, we implemented it using `torch.svd_lowrank()`, setting the number of iterations to 1.

`GenerativeAdapter` is able to generate the adaptors for prefixes of chunks simultaneously by processing the context chunks in parallel. The computation proceeds by processing the hidden states of each Transformer block for all context chunks layer by layer. Given the hidden states of $\Sigma(1), \ldots, \Sigma(t)$ from the $(l-1)$-th Transformer block, denoted by $H_{1:t}^{(l-1)}$, we first compute the accumulated outer product $S_{1:t}^{(l)}$ using Equation 3. We then normalize this outer product to obtain the additive matrix $W_{\Delta,1:t}^{(l)}$ using Equation 6, and finally get the output of the l-th Transformer block $H_{1:t}^{(l)}$ by the base model.

Pretraining. For Mistral-7B-Instruct (hereafter referred to as Mistral), we use a learning rate of 5×10^{-5}, and for Llama2-7B-Chat (Llama2), we use 1×10^{-4}. We apply a weight decay of 0.01 and no dropout. The adapter added to the base model are scaled by $1/16$ for Mistral and $1/8$ for Llama2. We employ a WarmupDecayLR learning rate scheduler with a 100-step warmup and use the Adam optimizer. The global batch size is set to 8. Pretraining the adapter generator on 1B tokens takes approximately 20 hours using 8 NVIDIA H100 GPUs.

Instruction Tuning. For instruction tuning, we largely follow the same configurations as in pretraining, with some adjustments. We set the learning rate to 5×10^{-5} for both Mistral and Llama2 models. We train the models for 2 epochs and use a batch size of 32.

D Experiment Details

D.1 Document-based Question Answering with Varying Context Length

The factual knowledge stored in the parameters of a LM remains static after pretraining. Here, we consider the scenario where the model needs to adapt to new knowledge based on documents. After adaptation, it is expected to correctly answer information-seeking questions about these documents.

Setup and Baselines To evaluate the fact recall ability of the adapted model, we use two question answering (QA) datasets, SQuAD [31] and StreamingQA [24], where each test case consists of a passage and a corresponding question about some information from that passage. To analyze the impact of context length on performance, we conduct an evaluation using contexts of varying lengths.

We divide the documents in the corresponding test set evenly into groups, with each group having an average length of k tokens ($k \in \{512, 1K, 2K, 4K, 8K, 16K, 32K\}$). Thus, the model should contextualize on the article in each group and evaluate fact recall by the question associated with the articles. The QA accuracy is evaluated by comparing the generated output with the gold answer for all questions associated with the documents within the group. Following [31], F1 score is used as the metric for evaluation.

We also analyze the computational and storage requirements of `GenerativeAdapter`, which comprises three phases: general-purpose pretraining, contextualization, and inference. The generator is pretrained once and can subsequently be used for any task. During the contextualization phase, `GenerativeAdapter` encodes the context into an adapter with a single forward pass. In the inference phase, the adapted model generates responses based on the input. Beyond the LM parameters, the extra storage required includes the parameters of the generative adapter.

Here, we consider both full parameter fine-tuning and full context prompting using the same base model as baselines. For fine-tuning, we consider two variants. The first approach, *supervised fine-tuning* (SFT), trains the base model exclusively on a training set of question-answer pairs sourced from articles distinct from those in the test set. The second variant, known as *continual pretraining* (CPT), involves first training the base model on all documents in the test set, followed by further adaptation through SFT using the the training set of question-answer pairs. During inference, we evaluate the fine-tuned model in a closed-book manner, *i.e.,* the model is tasked with directly producing the answer to a given question. For prompting, we simply concatenate all documents in the group as a single context and prompt the base model to respond accordingly. Specifically, for Llama2-7B-Chat, if the context length exceeds the maximum limit of 4K tokens, we truncate the prompt to include only the last 4K tokens. For `GenerativeAdapter`, we create an adapted model for each document group, which is similar to how the context is encoded as prompting. After that, the adapted model is asked to answer the question again in a closed-book fashion, akin to fine-tuning.

We set up experiments for document-based question answering (QA) using both supervised fine-tuning and continuous pretraining. For supervised fine-tuning on question-answer pairs, we train on the training split of each dataset, evaluate on a validation set, and employ early stopping when the validation loss increases. We use a learning rate of 1×10^{-5} and a global batch size of 64. For continuous pretraining, we train for 8 epochs using the log-likelihood of the document as the training loss, with learning rates of 1×10^{-5} for Mistral and 3×10^{-5} for Llama2. Each passage is treated as a training sample, and we use a global batch size of 16.

For closed-book prompting and in-context prompting, we apply an instruction template to encourage the model to generate a short answer. The prompts are shown in Figure 7.

D.2 In-Context Learning with varying in-context examples

In the prompting paradigm, one emerging ability of pretrained LMs is that they can perform a task with a few task-specific input-output examples as context on unseen cases, also known as in-context learning [4]. Here, we are interested to see whether `GenerativeAdapter` can provide further benefits in enhancing the base LM's in-context learning ability.

We conduct experiments using MetaICL [27], consisting of 26 test tasks. We also ensure that none of these test tasks were seen during the training of adapter generator. For each task, we use 1, 2, 4, 8, and 16 demonstrations randomly sampled from the corresponding development split following the MetaICL evaluate pipeline. To reduce evaluation variance, we repeat the sampling process five times

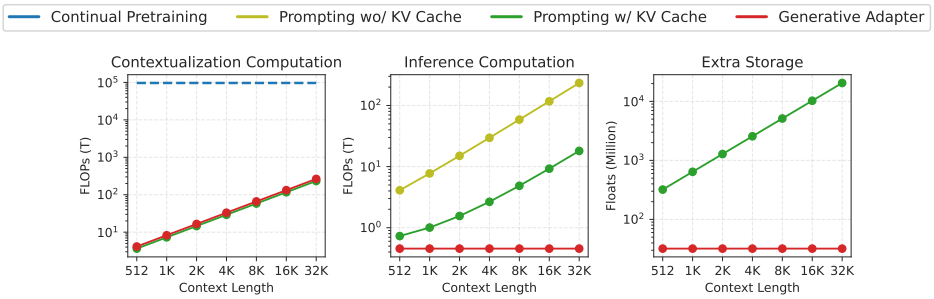


Figure 4: Computation and storage requirements for `GenerativeAdapter` and baseline methods on StreamingQA. For `GenerativeAdapter`, the context is converted into an adaptor during contextualization and then stored for inference. For the prompting method, the key-value (KV) cache can be generated during contextualization and reused during inference.

for each few-shot setting. We report separate average accuracy for classification and non-classification tasks. For classification tasks, achieving high accuracy requires the model to learn both the candidate options and the input-output relationships from the provided examples. For non-classification tasks, the model also needs to learn the output style.

We explore in-context learning using both fine-tuning and prompting methods. For fine-tuning, we conduct task-specific fine-tuning on 16 samples for each dataset. We use a learning rate of 5×10^{-6} for Mistral and 1×10^{-5} for Llama2. A validation set of 16 samples, disjoint from the training set, is collected from the same dataset. We train the model for a maximum of 40 epochs, employing early stopping if the validation loss increases for three consecutive epochs.

For in-context prompting, we observe that omitting additional instructions yields better performance for Mistral, whereas adding an instruction template improves performance for Llama2. The prompts are shown Figure 7.

E Additional Results: Personalization

Using LMs to analyze users' behaviours and memorize their preferences is the key to unlocking a tailored and engaging user experience, *i.e.,* personalized LMs. Towards this goal, we focus on evaluating the LM's ability to memorize user information in conversations.

Setup and Baselines We use the Multi-Session Conversation (MSC) dataset [42] for our experiments, following [28]. Each test case comprises a multi-session human-human conversation between two participants, along with a question regarding information mentioned within the conversation. The average length of the conversational context is 2.5K tokens, which makes it inefficient to prompt the model repeatedly with the entire conversation history for the same user. Similar to document-based QA (§D.1), we evaluate the model quality using the F1 score by comparing the generated answers to the ground truth. We also report computation and memory costs. Here, we use Mistral-7B-Instruct as the base LM.

As baselines, we include both closed-book and full-conversation prompting based on the base LM, where the former involves random guesses and the latter incurs higher computation and memory costs by storing the entire long conversation. We also include the state-of-the-art prompt compression method, UltraGist [44], which reduces the context into fewer token embeddings, thereby saving computation and memory costs.

Results The results on MSC are summarized in Table 1. As expected, the closed-book approach, which does not memorize any user information performs very poorly. In contrast, methods that utilize proper user conversations as context can accurately recall user information, achieving reasonable answer accuracy. Although using the entire conversation leads to better accuracy, full conversation prompting incurs significant computation and storage costs, *i.e.,* 4x those of `GenerativeAdapter`. Such costs are highly undesirable for personalizing LMs for individual users, especially since most computations occur on edge devices without power GPUs. Comparing to UltraGist at the same level of storage cost (compressed into 512 tokens), `GenerativeAdapter` further reduces inference cost

Table 1: Performance comparison on MSC. A higher F1 indicates better performance, and lower inference computation and extra storage costs are preferable. For Ultragist [44], fewer compressed tokens (noted in parentheses) correspond to lower computation and memory costs.

Model	F1	Inference Computation (TFLOPS)	Extra Storage (M floats)
Closed-book	8.1	0.505	0
Full-conversation Prompting	66.0	2.059	128+
Ultragist (64 Tokens)	26.5	0.514	4
Ultragist (128 Tokens)	32.2	0.552	8
Ultragist (256 Tokens)	38.3	0.627	16
Ultragist (512 Tokens)	40.8	0.772	32
Ultragist (1K Tokens)	44.4	1.067	64
Ultragist (2K Tokens)	42.4	1.658	128
Generative Adapter	40.2	0.505	32

without performance drop. In real world scenarios with many queries from the same user, the benefits of our method are even more pronounced.

F Additional Results: Ablation Study

Here, we examine how different design choices with `GenerativeAdapter` affect model performance. Specifically, we train adapter generators for Mistral-7B-Instruct under various configurations and evaluate them using two metrics—reconstruction perplexity and completion perplexity—on a validation set drawn from the same distribution as the pretraining corpus. As we observe in our preliminary study, the quality of the resulting adapter generator is highly correlated with these metrics. The results are presented in Table 2, where the default setting is described in §2.

Mix of both pretraining tasks is necessary. As we can see, relying solely on one task does not yield good perplexity on the validation set for both metrics. In particular, training with only reconstruction task results in a significant drop in completion perplexity, indicating a loss of general language modeling capabilities and overfitting to memorization. Thus, the completion task acts as data regularization for `GenerativeAdapter`, enabling the model to distill contexts into generated adapters and effectively utilize them in future predictions.

SVD is a more effective normalization. We explore an alternative normalization for Equation 6 using the Frobenius norm, defined as $\text{norm}(M) = M/\|M\|_F$, where $\|M\|_F = \sqrt{\sum_{i,j} M_{ij}^2}$. Compared to SVD used in our default setting, the Frobenius norm is computationally simple and helps bound the matrix scale. However, our results indicate that the Frobenius norm is not as effective. Observing substantial disparities in the scale of singular values, we hypothesize that applying the Frobenius norm may unnecessarily shrink certain directions, reducing the model's expressiveness.

More update parameters lead to better performance. By default, we add the adapter into the output projection layer of the attention module. We also experiment with adding the adapter to the down projection layer within the feedforward module, which introduces more update parameters (3x those of default). We observe that placing the adapter on the feedforward layer indeed leads to slightly improved reconstruction and completion perplexities. Due to computational constraints, a more thorough exploration was not feasible and we leave this for future investigation.

G Related Works

Fast Weights: Our proposed method is closely related to the idea of "fast weights" [11, 2, 35], which makes the model weights being adaptive to the model input. Context-dependent fast weight programmers (FWPs) introduced by **(author?)** [36, 37] use a slow network with slow weights to reprogram the fast weights of the corresponding fast network. [35] point out that self-attention without softmax and other linear Transformer variants [41, 18, 6, 29] can be viewed as FWPs.

[7] propose fast weight layers which are added on top of the Transformer model after the last attention layer for language modeling. Different from previous work mainly focusing on specific tasks, our goal

Table 2: The validation set perplexity of the pretrained model under different design choices.

Factor	Setting	Reconstruction Perplexity	Completion Perplexity
-	Default	1.75	7.40
Pretraining Task	Reconstruction	1.75	34.34
	Completion	6.38	6.71
Normalization	Frobenius	7.72	7.32
Module	Feedforward	1.68	7.26

Table 3: Statistics of data used in the instruction tuning.

Type	Dataset	#Docs	#Instructions	Context len	Instruction len	Response len
Question Answering	COQA [32]	1798	57.2	1083.5	5.5	2.7
	DROP [8]	1379	38.4	848.6	11.0	1.4
	NarrativeQA [20]	1047	29.4	574.5	8.5	4.4
	PubMedQA [17]	1000	1.0	200.2	12.9	40.7
	Quail [33]	560	16.2	332.7	8.7	4.9
	MS MARCO [3]	4832	16.5	1152.1	6.0	14.0
In-context Learning	MetaICL [27]	11888	3.5	1776.8	84.3	2.9
Instruction Following	BookSum [21]	2914	1.0	1158.6	7.0	205.3
	PwC [9]	13102	12.4	348.1	10.3	23.3

is to enhance frozen pretrained LMs with fast associative memory for general language processing. Instead of using a slow network to program a separate fast model, our method can be viewed as a self-programming model, *i.e.,* context encoded by the base LM is used to update the base LM itself.

Adapting LMs via Meta-Learning: One line of work focuses on adapting pre-trained LMs for an online stream of documents using meta-learning. Observing that naively fine-tuning on the documents using the negative log-likelihood loss is not effective for downstream question answering, [14] propose context-aware meta-learned loss scaling to re-weight the loss for individual tokens based on their importance during the online fine-tuning. [39] use a meta-learned amortization network to directly predict parameter efficient fine-tuning modulations of the base LM for individual context documents. The modulations are then aggregated into a single output for downstream question answering. Unlike those methods that typically require a nested training loop, our adapter generator augments pretrained LMs and our model can be trained in an end-to-end fashion with self-supervised objectives.

Parameter-Efficient Fine-Tuning (PEFT): `GenerativeAdapter` employs a low-rank adapter akin to LoRA [13], which was originally designed for PEFT. Several derivatives of LoRA exist such as AdaLoRA [45] and DoRA [25], along with various other PEFT strategies such as serial adapters [12] and prefix tuning [22]. A thorough survey of PEFT methods is presented by [10]. Most work focuses on task-specific fine-tuning scenarios. Instead, `GenerativeAdapter` is a general LM and does not require a downstream dataset for adaptation.

Table 4: Training and test datasets of MetaICL.

Train	piqa, hate_speech_offensive, google_wellformed_query, social_i_qa, circa, quoref, glue-sst2, scitail, emo, cosmos_qa, freebase_qa, ag_news, art, paws, kilt_ay2, glue-qnli, quail, ade_corpus_v2-classification, sciq, hatexplain, emotion, glue-qqp, kilt_fever, kilt_nq, dbpedia_14, kilt_zsre, hellaswag, squadwith_context, hotpot_qa, glue-mnli, ropes, squad-no_context, kilt_hotpotqa, discovery, superglue-record, race-middle, race-high, lama-trex, swag, gigaword, amazon_polarity, biomrc, tab_fact, tweet_eval-emoji, tweet_eval-offensive, tweet_eval-sentiment, tweet_qa, imdb, lama-conceptnet, liar, anli, wiki_qa, kilt_trex, wikisql, wino_grande, wiqa, search_qa, xsum, yahoo_answers_topics, yelp_polarity, yelp_review_full
Test	quarel, financial_phrasebank, openbookqa, codah, qasc, glue-mrpc, dream, sick, commonsense_qa, medical_questions_pairs, quartz-with_knowledge, poem_sentiment, quartz-no_knowledge, glue-wnli, climate_fever, ethos-national_origin, ethos-race, ethos-religion, ai2_arc, hate_speech18, glue-rte, supergluecb, superglue-copa, tweet_eval-hate, tweet_eval-stance_atheism, tweet_eval-stance_feminist

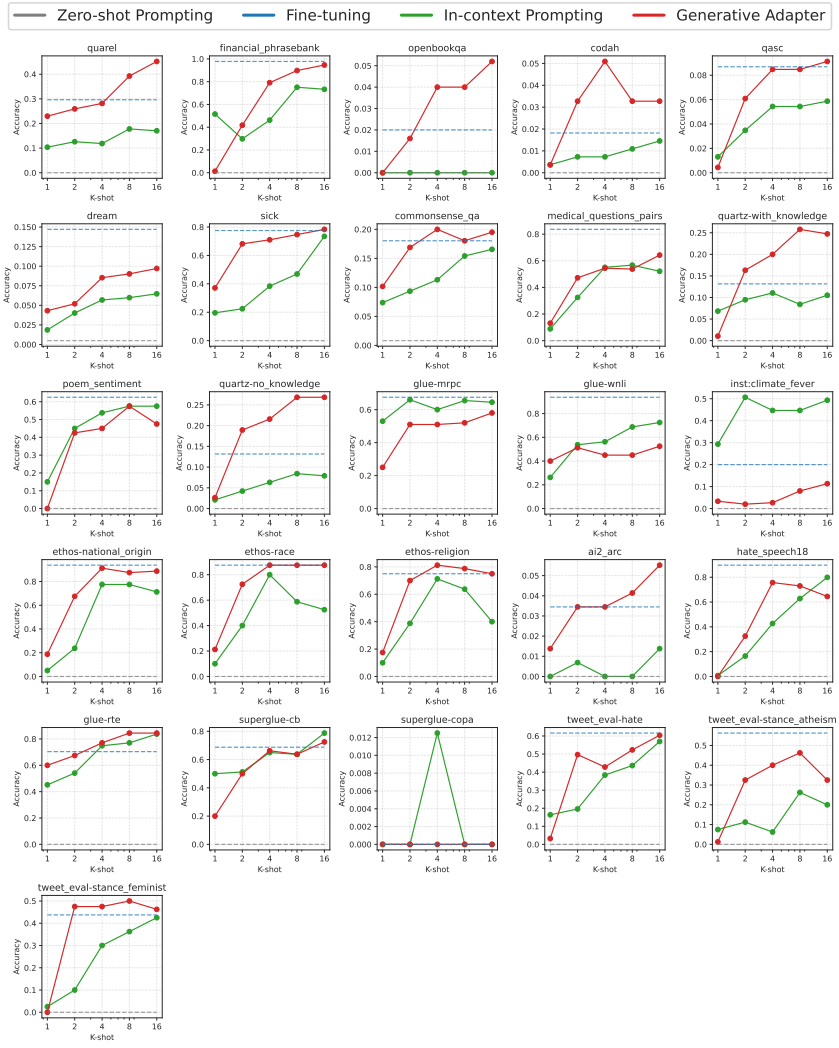


Figure 5: In-context learning evaluation of `GenerativeAdapter`, based on Llama2-7B-Chat, across 26 test datasets from MetaICL.

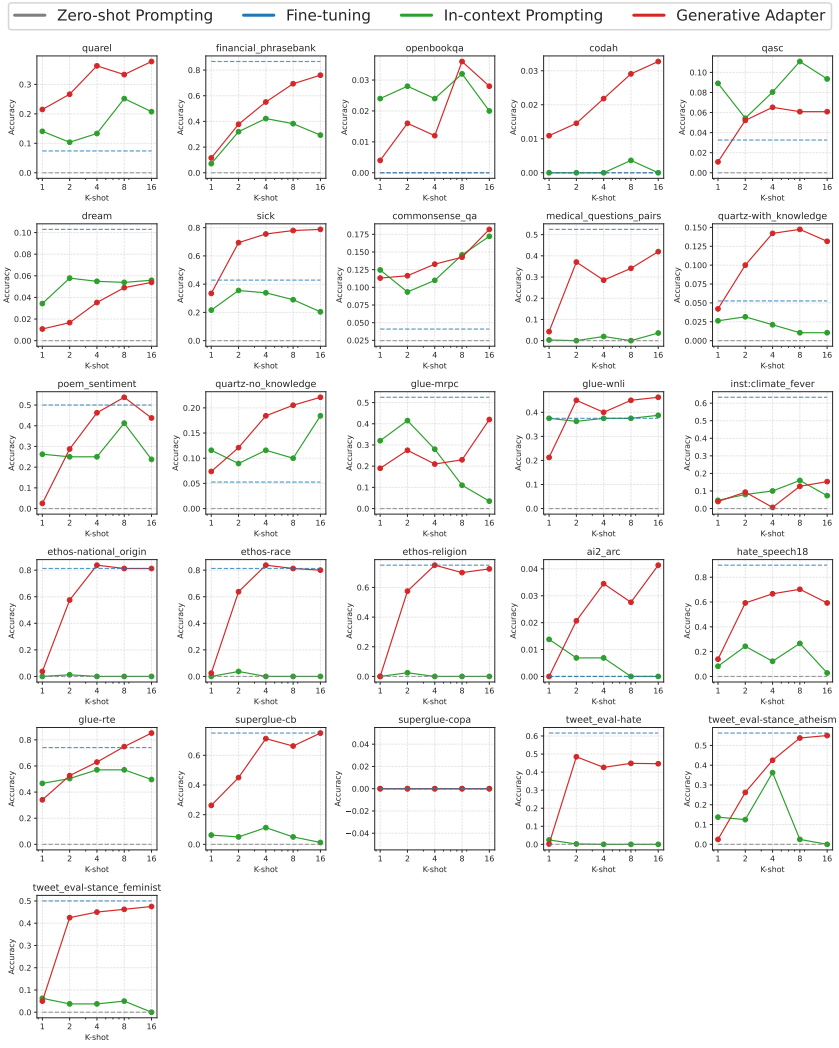


Figure 6: In-context learning evaluation of `GenerativeAdapter`, based on Mistral-7B-Instruct, across 26 test datasets from MetaICL.

Model	Dataset	Methods	F1						
			512	1K	2K	4K	8K	16K	32K
Mistral	SQuAD	Zero-Shot Prompting	10.8						
		Supervised Fine-tuning	20.7						
		Continuous Pretraining	30.0						
		In-context Prompting	45.4	44.9	43.6	42.6	42.5	38.6	35.1
		GenerativeAdapter	48.8	43.0	39.9	35.9	33.8	30.3	28.0
	StreamingQA	Zero-Shot Prompting	13.6						
		Supervised Fine-tuning	19.5						
		Continuous Pretraining	22.2						
		In-context Prompting	47.2	48.7	48.1	48.7	48.0	46.0	39.3
		GenerativeAdapter	51.5	49.3	44.7	40.9	36.7	32.7	32.0
LLama2	SQuAD	Zero-Shot Prompting	14.6						
		Supervised Fine-tuning	20.7						
		Continuous Pretraining	23.9						
		In-context Prompting	64.8	60.6	55.4	44.9	25.2	9.6	6.4
		GenerativeAdapter	36.2	32.5	31.0	28.9	28.2	24.9	23.6
	StreamingQA	Zero-Shot Prompting	18.0						
		Supervised Fine-tuning	18.9						
		Continuous Pretraining	20.5						
		In-context Prompting	61.2	61.3	58.0	46.8	27.8	17.5	11.6
		GenerativeAdapter	42.9	37.8	34.4	32.6	28.7	26.0	25.7

Table 5: All results of the QA accuracy on SQuAD and StreamingQA.

Prompting for Document-based Question Answering

{Context}

Instruction: Answer the question based on the context above. Respond with a short phrase only. Keep the answer short and concise, without any explanation or additional words

Question: {Question}
Answer:

Prompting for MetaICL

Input: {demo input}
Output: {demo output}
{ . . . k-shot demonstrations . . . }

Instruction: Based on the demonstration above, provide a short and concise answer, without any explanation or additional words.

Input: {input}
Output:

Figure 7: Prompts used in the document-based QA and in-context learning evaluation.