36. In-Context Learning in LLMs via Test-Time Training

Matvey Kairov, Daniil Sergeev

AIRI Summer School 2025

Abstract

We propose a hybrid framework that integrates gradient-based meta-learning and test-time adaptation to enable efficient in-context memorization and question answering. A compact memory module is meta-trained to acquire efficient initializations and hyperparameters for the inner loop and then updated at inference via a small number of gradient steps on a self-supervised next-token objective. At query time, the adapted memory serves as a compressed representation of contextual information, enabling fact retrieval through an attention mechanism. Empirical evaluation on a synthetic associative retrieval benchmark demonstrates our model's ability to retrieve information acquired through meta-learning to answer questions based on the provided context while reducing computational overhead.

1 Introduction

Rapid adaptation to novel tasks with limited data remains a fundamental challenge in machine learning. Model-Agnostic Meta-Learning, proposed by Finn et al. [1] addresses this challenge by optimizing a model initialization θ such that a small number of gradient descent steps on each task's loss yield strong performance, as seen in Figure 1. Despite its generality and empirical success across classification, regression, and reinforcement-learning domains, MAML requires explicit fine-tuning of the entire parameter set at test time.

Test-Time Training introduced by Sun et al. [2] extends the adaptation paradigm by embedding a self-supervised learning loop within the inference process. TTT updates a small auxiliary network — either linear or multilayer perceptron — using an unsupervised objective on the incoming data sequence, thereby improving the model's fit to the test distribution without full retraining.

In this work, we combine these approaches by meta-training a low-dimensional memory module and performing targeted parameter updates only within this module at inference. During meta-training, the memory module is optimized alongside the base model to facilitate rapid memorization. At test time, we apply a few gradient steps on a next-token prediction loss to adapt the memory

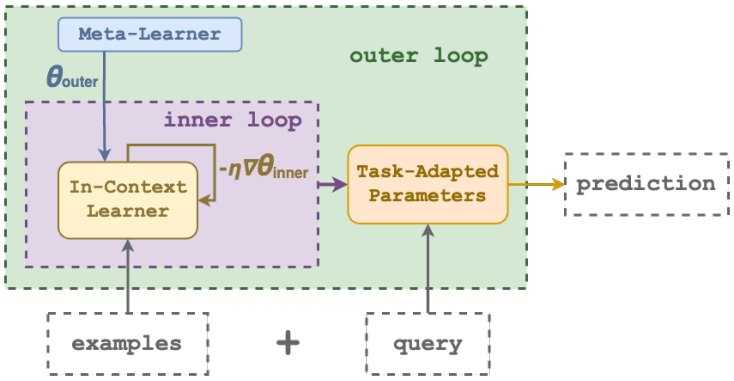

Figure 1: A general scheme of Model-Agnostic Meta-Learning.

to the new context. The adapted memory, when queried by an attention mechanism, yields facts with a linear complexity, significantly reducing computational overhead. We evaluate the proposed method on a synthetic associative retrieval dataset in order to test its memorization ability.

2 Methodology

In its simplest formulation, test-time training for memorization can be performed in the following way. Given some input context, which we expect to get an answer to, we perform a number of gradient descent steps in order to memorize this context at test time. In order to make the operation less computationally expensive and more scalable, we optimize only a small set of parameters, which can include a trainable prompt, a low-rank adaptation [3] of the backbone model or some combination of the two, to memorize the context via a standard language modeling task. In this task, the model is required to predict a token based on the previous one.

Kuratov et al. [4] have explored the capacities of input tokens for LLMs and found out that even a single token may be used to restore a sequence hundreds of tokens in length. This can be done by a process known as P-Tuning [5], where a set of input embeddings for a model are optimized in order to reproduce some desired behavior when appended to the beginning of the input sequence, while the parameters of the model itself are kept in touch. The same logic can be applied to our task, where we aim to memorize a sequence as best as we can in order to quickly retrieve facts from the acquired compressed representation, but is not limited to strictly P-Tuning: trainable weights can also be used for memorization.

A general framework for test-time training memorization is outlined in 2. Here, context is memorized using gradient descent updates in an inner loop, after which information is retrieved from it in order to answer a query. A

Transformer model (or any other LLM) is used to both aid memorization and retrieve information from memory, with its weights updated only in the outer loop based on its answer to the query.

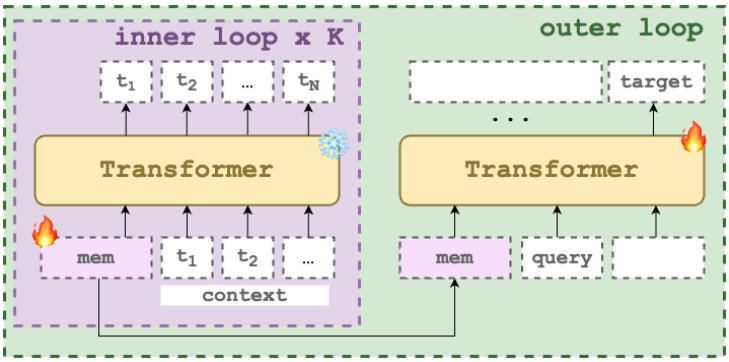

Figure 2: A baseline setup for LLM Test-Time Training.

We choose this method as our baseline for test-time training memorization.

2.1 NeuralMemory Block

Inspired by the ideas proposed by Behrouz et al. in Titans [6], we introduce a NeuralMemory block 3. It receives two inputs during the forward pass: slots (into which the memory will be written) and a context embedding. The core idea is to iteratively write contextual information into these slots using cross-attention, where the queries (q) are derived from the memory slots, and the keys (k) and values (v) are projections of the context embedding. After writing information into the memory slots through several inner-loop steps, we pass these slots into the outer loop and concatenate them with the query embedding to generate the model's output. The outer loop trains the model to retrieve relevant information from the memory slots for accurate generation. A stateless optimizer is used for the inner loop (to train the memory slots), while a stateful optimizer is used for the outer loop (to update the LLM parameters and memory slots)

2.2 MemoryMLP

In the original RMT, learnable memory tokens serve as memory. We extended this by introducing a trainable MLP, or MemoryMLP, and using its parameters as an additional memory mechanism 4.

During the inner loop, we compute a context embedding and average it across the sequence length to produce a single vector. This averaged embedding is then fed into the MemoryMLP, which outputs vectors of the same dimensionality as the memory tokens. These outputs are added to the existing memory tokens. The idea is for the MemoryMLP to learn to store contextual information both

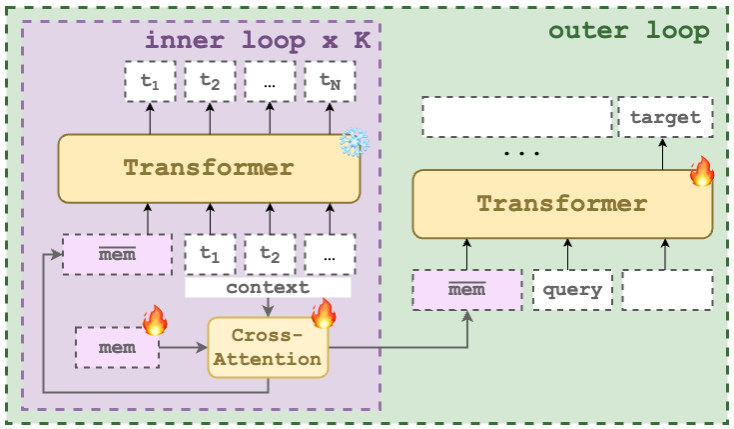

Figure 3: Test-Time Training with a Neural Memory block.

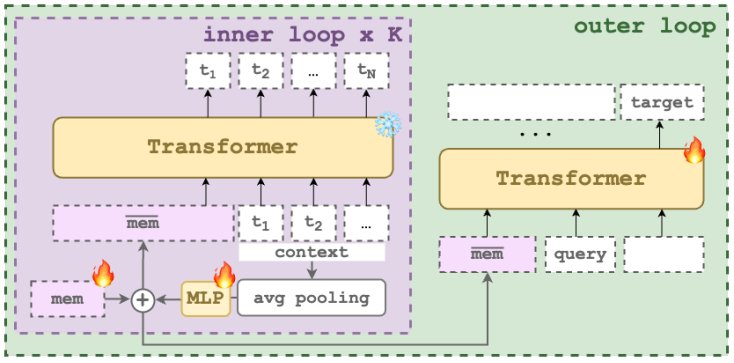

Figure 4: Test-Time Training with a Memory MLP block.

in the memory tokens and in its own parameters, enabling improved generation. The combined memory tokens are then passed into the outer loop, where they are concatenated with the query embedding to produce the final output. At this stage, the model learns to "read" from memory and extract the necessary information for generation. A stateless optimizer is used in the inner loop (to update the parameters of MemoryMLP and the memory tokens), while a stateful optimizer is used in the outer loop (to update the LLM parameters and the combined memory tokens)

2.3 Recurrent Memory Transformer with Test-Time Training

Recurrent Memory Transformer, proposed by Bulatov et al. [7], is an augmentation for Transformer models that extends their context size by segmenting

sequences and processing them recurrently, resulting in linear scaling with input size. Memory, implemented as special memory tokens, is processed alongside segment tokens to enable the Transformer to access information from previous segments.

For the time step i and segment H_i^0, the recurrent step is performed as follows:

$$\tilde{H}_i^0 = \left[H_i^{\text{mem}} \circ H_i^0 \circ H_i^{\text{mem}}\right],$$
$$\bar{H}_i^N = \text{Transformer}\left(\tilde{H}_i^0\right),$$
$$\left[\bar{H}_i^{\text{mem}\prime} \circ H_i^N \circ \bar{H}_i^{\text{mem}}\right] := \bar{H}_i^N,$$

where N is the number of Transformer layers.

We introduce the Recurrent Memory Transformer with Test-Time Training (RMT4), which integrates RMT with Test-Time Training by splitting the context into segments and memorizing each segment consecutively, replacing the original memory update mechanism with our gradient descent-based approach (Figure 5). Similar to previous propositions, memorization is performed using prompt-tuning, but memories accumulate in the memory tokens between segments.

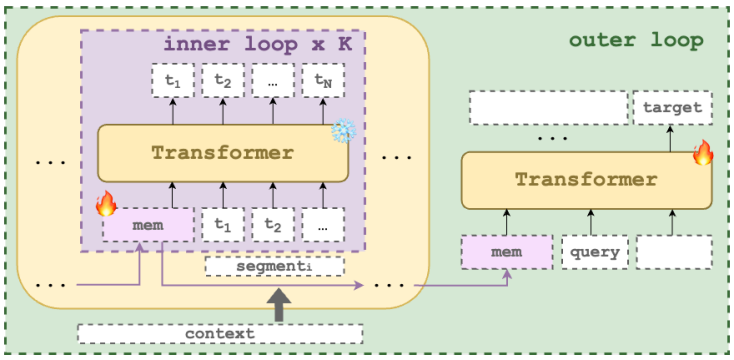

Figure 5: Recurrent Memory Transformer with Test-Time Training (RMT4). Each segment is memorized separately, with gradient memory updates accumulating in memory tokens.

2.4 Parallelizing memory updates in RMT4

Associative Recurrent Memory Transformer, proposed by Rodkin et al. [8], augments the original RMT by introducing layer-wise associative memory updates. This allows this architecture to not only outperform multiple other architectures on long-context language modeling tasks, but also be partially parallelizable by virtue of its granular memory updates. Diagonal Batching, introduced by

Sivtsov et al. [9], exploits this style of memory updates by processing the next segments of the sequence without having to wait for the previous segment to finish propagating through the model. Memory updates on layer i rely on the updated memories from the respective layer on the previous segment as well as the output of the previous layer, so, given both of those, they can begin to process the next segment right away. This allows for a significant computational speedup, along with a more flexible mechanism for memory updates.

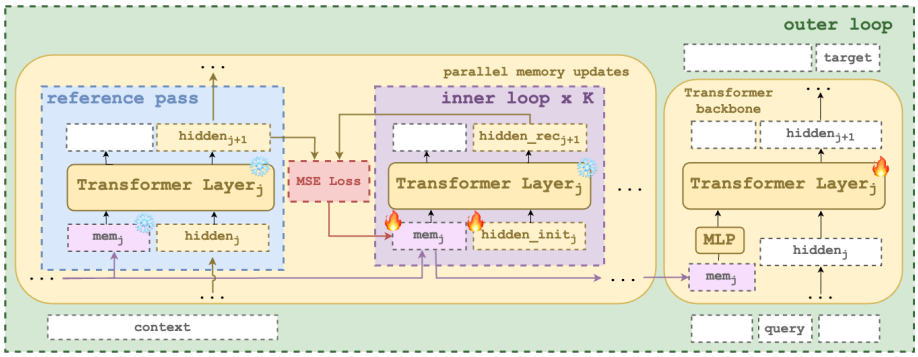

Figure 6: Parallel RMT4. Memory updates are separated for Transformer layers, learning to reconstruct their respective hidden states.

One challenge that arose from this idea was its incompatibility with our original method for context memorization. Since hidden states in between layers do not directly correspond with next-token prediction, we were unable to use a cross-entropy loss function between layer outputs and expected hidden states. Instead, we propose the following solution: instead of cross-entropy between expected and predicted hidden states, we train the memory tokens to reconstruct the hidden states expected as the output of the layer. To aid in this reconstruction, we prepend our test-time updated memory tokens to a set of trainable global memory tokens, which act as initializations for hidden states. We then update the memory tokens based on a mean squared error loss between real hidden states, acquired from concatenating the old memory state with input hidden states and passing them through the layer as is, with our "reconstructed" hidden states, which correspond to the global memory in the second input sequence. We named this method Parallel RMT4 (Figure 6).

This approach allows for parallelization not only through diagonal batching, but also by continuing the model forward without waiting for the test-time backpropagation.

2.5 Experimental setting

We used small versions of Pythia [10] and LLaMa-3.1 [11] as the backbone Transformer for our proposed architectures with 4 layers, 4 attention heads on each layer and an embedding dimension of 128. While we haven't settled on

an optimal set of hyperparameters, we used 8 memory tokens, an inner loop learning rate of 1, and 2 to 5 steps of gradient descent in the inner loop.

In order to test our methods, we used the associative retrieval task, similar to the one utilized in [8]. In this dataset, tasks are generated in the following way: unique pairs of keys and values are generated from random tokens, after which they are concatenated with some random noise in between pairs. An example of this noisy associative retrieval task can be seen in Figure 7.

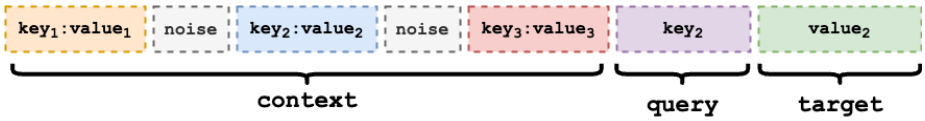

Figure 7: An example of a noisy associative retrieval task. In the context part of the sequence, key-value pairs are separated by noise. The model has to retrieve the target based on the provided query, which are, respectively, a value and a key from one of the pairs provided in the context.

3 Results

Both datasets were generated with the following features:

- each key and value consist of 4 tokens;
- each segment is between 16 and 32 tokens long.

The results of the training of our models is provided in Table 1.

	1 pair, 1 segment	**2 pairs, 2 segments**
Baseline TTT	100.0	100.0
Neural Memory	99.9	72.1
Memory MLP	94.7	4.5
RMT4	99.0	0.0
Parallel RMT4	6.0	0.0

Table 1: Accuracy on associative retrieval tasks, %.

Neural Memory, Memory MLP and RMT4 have all converged or came close to convergence on the 1-pair, 1-segment task. Neural Memory began converging on the 2-pair dataset, but did not manage to converge fully in a reasonable time. Parallel RMT4 did not converge on any task — possibly, due to challenges with implementation.

4 Conclusion

We have proposed 4 architectures for the test-time training memorization task: Neural Memory, Memory MLP RMT4 and Parallel RMT4. We trained these models to solve the associative retrieval task with 1 and 2 pairs in the context. None of the proposed methods has managed to overcome the baseline test-time training memorization method. Neural Memory has shown the best performance out of the proposed methods. Given the lack of hyperparameter tuning, this architecture has the potential to overcome our baseline method.

5 Limitations and Future Work

While the proposed augmentations to language models show promising results on the tasks of context memorization and retrieval, there are several key limitations to our work that require further research. First, our experiments include fine-tuning models to solve the proposed associative retrieval task, which, while representative of the model's ability to memorize information at test time and access those memories, may not fully indicate its potential for reasoning based on memory acquired at test time. Testing our approach at longer associative retrieval tasks and other question-answering tasks such as BABILong [12] could offer important insights into the proposed models' memorization ability.

Furthermore, we are yet to explore the appropriate hyperparameters for each of the models. While the current experimental setting allows for some of the models to converge on simpler tasks, they still struggle with memorizing longer sequences.

We have outlined the following directions for perspective research on this topic:

1. Fine-tune our models on more complicated question-answering tasks such as those in BABILong.
2. Search the hyperparameter space for each model to find the optimal combination of hyperparameters for training.
3. Perform a full ablation study, going over all proposed augmentations to find out the effects introduced by each one of them.
4. Explore further approaches to memorization such as training a Low-Rank Adaptation setup [3] in the inner loop.
5. Work out a way to train the Parallel RMT with Test-Time Training, and further optimize it using Diagonal Batching.

References

[1] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[2] Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): Rnns with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024.

[3] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

[4] Yuri Kuratov, Mikhail Arkhipov, Aydar Bulatov, and Mikhail Burtsev. Cramming 1568 tokens into a single vector and back again: Exploring the limits of embedding space capacity. *arXiv preprint arXiv:2502.13063*, 2025.

[5] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. Gpt understands, too. *AI Open*, 5:208–215, 2024.

[6] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*, 2024.

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

[8] Ivan Rodkin, Yuri Kuratov, Aydar Bulatov, and Mikhail Burtsev. Associative recurrent memory transformer. *arXiv preprint arXiv:2407.04841*, 2024.

[9] Danil Sivtsov, Ivan Rodkin, Gleb Kuzmin, Yuri Kuratov, and Ivan Oseledets. Diagonal batching unlocks parallelism in recurrent memory transformers for long contexts. *arXiv preprint arXiv:2506.05229*, 2025.

[10] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR, 2023.

[11] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[12] Yury Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack. *Advances in Neural Information Processing Systems*, 37:106519–106554, 2024.