# Toward generalizable learning of all (linear) first-order methods via memory augmented Transformers

Sanchayan Dutta
UC Davis, Davis CA, USA
Suvrit Sra
TU Munich, Garching, Germany

dutta@ucdavis.edu

s.sra@tum.de

#### **Abstract**

We show that memory-augmented Transformers can implement the *entire class of linear first-order methods* (LFOMs), a class that contains gradient descent (GD) and more advanced methods such as conjugate gradient descent (CGD), momentum methods and all other variants that linearly combine past gradients. Building on prior work that studies how Transformers simulate GD, we provide theoretical and empirical evidence that memory-augmented Transformers can learn more advanced algorithms. We then take a first step toward turning the learned algorithms into actually usable methods by developing a mixture-of-experts (MoE) approach for *test-time adaptation* to out-of-distribution (OOD) samples. Lastly, we show that LFOMs can themselves be treated as learnable algorithms, whose parameters can be learned from data to attain strong performance.

## 1 Introduction

In-context learning (ICL) allows large language models (LLMs) to generate contextually appropriate outputs based solely on examples and queries provided in a prompt (Brown, 2020; Liu et al., 2021; Lu et al., 2021; Wei et al., 2022; Wu et al., 2022). This remarkable ability has spurred research into understanding how Transformers can implement algorithms (Achiam et al., 2023; Touvron et al., 2023), with recent studies focusing on they can simulate optimization algorithms (Dai et al., 2022; Von Oswald et al., 2023a; Garg et al., 2022; Akyürek et al., 2022). Transformers have been shown to implement gradient-based optimization during their forward pass, such as preconditioned gradient descent for linear regression tasks (Dai et al., 2022; Mahankali et al., 2023; Ahn et al., 2024).

Other recent work shows that Transformers can also learn more advanced optimization methods, e.g., Fu et al. (2023) show that Transformers exhibit convergence rates comparable to Newton's Method for ICL on linear regression. Vladymyrov et al. (2024) prove that Transformers can learn a variant of gradient descent that approximates second-order methods, such as GD<sup>++</sup>, achieving convergence rates similar to Newton's method. These findings lead to the central question of our paper:

Can Transformers efficiently learn more advanced gradient-based optimization methods?

We address this question by investigating the representational power of Transformers as *algorithm learners*, and with it, contributing to the topic of machine driven algorithm discovery. More

specifically, we focus on learning all gradient-based algorithms obtainable by linearly combining past gradients, known as *Linear First-Order Methods (LFOMs)* (Goh, 2017), wherein (k + 1)-st iterate is

$$w^{k+1} = w^0 + \sum_{i=0}^k \Gamma_i^k \nabla f(w^i), \tag{1.1}$$

and where  $\{\Gamma_i^k\}_{i=0}^k$  are diagonal matrices. Iteration (1.1) is quite general: it includes as special cases standard methods such as gradient descent (GD), momentum GD, Nesterov's accelerated gradient, conjugate gradient, and in a stochastic version, AdaGrad, ADAM, among others.

Toward showing how Transformers can (efficiently) capture all LFOMs, our key insight is to consider memory-augmented Transformers, known as *Memformers* (Wu et al., 2020; Xu et al., 2021), which retain intermediate attention values across layers. This memory enables Memformers to store past gradients, and facilitates efficient mimicking of first-order methods such as conjugate gradient descent and momentum methods, that also use past gradients.

One may ask: *What does it mean to "learn" an optimization algorithm?* For us, it refers to two key aspects:

- 1. *Expressivity*. The Memformer can perform iterations of LFOMs in its forward pass, under a suitable choice of parameters. Thus, the architecture and parameterization are *sufficiently expressive* to simulate LFOMs.
- 2. *Trainability.* The Memformer's parameters can be trained on random linear regression tasks. Then, using these learned parameters (which are shared across all in-context data samples), the Memformer can execute "CGD-like" and "LFOM-like" iterations during a forward pass. (cf. Section 3)

While far from unconditional learning of GD, the above two aspects of learning follow a large body of work on how Transformers learn GD in context (Garg et al., 2022; Akyürek et al., 2022; Von Oswald et al., 2023a; Ahn et al., 2024; Zhang et al., 2024). Inspired by these works, and extending the work of Ahn et al. (2024), we study "learning" by analyzing the loss landscape for memory-augmented *Linear Transformers* that omit softmax activations (Schlag et al., 2021; Von Oswald et al., 2023a; Ahn et al., 2024).

#### 1.1 Main Contributions

- 1. Theoretical justification that Memformers can implement LFOM iterations, including CGD. We provide a rigorous theoretical framework showing that Memformers, when trained on linear regression tasks, can be configured to perform iterations of any desired LFOM in their forward pass.
- 2. Empirical evidence that Memformers learn optimization algorithms. Through extensive experiments, we demonstrate that a Memformer can learn LFOMs by training on random linear regression tasks. Thereafter, using its learned parameters, it can solve new regression problems using its forward pass. Notably, on several inputs such a Memformer performs competitively against (sometimes even outpeforming) CGD, Nesterov AGM, and momentum GD.

This finding is potentially significant because while CGD (and several LFOMs) adapts its step-size and conjugacy parameters *per sample*—which translates into 2*B* parameters  $\{\alpha, \gamma\}$  per iteration, for a batch of size *B*—a Memformer reuses one set of learned weights across the entire batch (see also Remark 3.3).

- 3. Enhanced performance through multi-headed attention and mixture-of-experts (MoE). We demonstrate that multi-headed attention improves both in-distribution and out-of-distribution (OOD) performance, with an MoE approach that enables each attention head to specialize in handling different data distributions. Additionally, we explore the performance of LFOMs when their parameters are learned directly from data, thus framing LFOMs as statistically learnable algorithms, and showcase their ability to generalize effectively across diverse in-context data samples.
- 4. Connections to test-time adaptation and relevance for LLMs. Beyond in-context adaptation, we investigate ways to update model parameters at test time to address shifting distributions, a topic with growing importance in large language models (LLMs). Recent work suggests that appropriately scaling test-time compute, rather than just model size, can yield stronger performance improvements (Snell et al., 2024), and we link these insights to our MoE-based Memformer approach (Section 4.2) to demonstrate how gating and memory preconditioners can adapt at inference time to handle various distribution shifts (Sun et al., 2020).

**Our objective.** We would like to emphasize that our main objective is to investigate the potential of Memformers to learn advanced optimization algorithms *in a general sense*, as well as their potential to adapt to OOD optimization problems. We are *not* advocating for Transformers as replacement for established optimization methods. Nevertheless, we hope that in the future, machine-discovered optimization algorithms surpass hand-designed methods such as GD or CGD, at least in an average-case / distributional sense.

#### 1.2 Related Work

**In-Context Learning.** The ability of Transformer models to perform in-context learning (ICL) has been extensively studied since its introduction by Brown (2020). Subsequent works have explored how these models adapt to new tasks without requiring parameter updates (Xie et al., 2021; Von Oswald et al., 2023b; Hahn and Goyal, 2023; Liu et al., 2021; Lu et al., 2021; Wei et al., 2022; Wu et al., 2022).

Gradient-Based Methods in Transformers. Garg et al. (2022) analyze the learning of GD within Transformers, particularly in the context of ICL for linear functions. Empirical studies (Garg et al., 2022; Akyürek et al., 2022; Von Oswald et al., 2023a) have shown that Transformers can learn GD after being trained on random linear regression tasks. Expanding on these results, Von Oswald et al. (2023a); Ahn et al. (2024) demonstrate that Transformers can implement preconditioned GD for solving linear regression problems presented in input prompts. Notably, these works, as well as ours, utilize Linear Transformers as discussed in (Schlag et al., 2021; Von Oswald et al., 2023a; Ahn et al., 2023).

**Higher-Order Optimization Methods in Transformers.** Transformers have also been shown to learn higher-order optimization techniques, such as Newton's method, expanding their capabilities beyond first-order methods (Fu et al., 2023; Giannou et al., 2024; Vladymyrov et al., 2024).

Memory-Augmented Transformers (Memformers). Memformers were introduced in (Wu et al., 2020; Xu et al., 2021). While significant progress has been made in understanding how Transformers can learn GD, their potential for learning more sophisticated methods remains largely unexplored. Our work addresses this gap and shows how Memformers can efficiently implement a wide range of advanced first-order and quasi-second-order methods.

## 2 Background and Problem Setup

We follow the setup of training Transformers on random instances of linear regression, following the prior works (Garg et al., 2022; Akyürek et al., 2022; Von Oswald et al., 2023a; Ahn et al., 2024). We largely follow the notation and formal setup of (Ahn et al., 2024), which we recall below.

## 2.1 Linear Transformers on Random Linear Regression

**Data Distribution.** Let  $\mathbf{x}(i) \in \mathbb{R}^d$  represent covariates drawn independently from a distribution  $\mathcal{D}_{\mathbf{X}}$ , and let  $\mathbf{w}^* \in \mathbb{R}^d$  be drawn from  $\mathcal{D}_{\mathbf{W}}$ . The matrix of covariates  $\mathbf{X} \in \mathbb{R}^{(n+1)\times d}$  contains rows  $\mathbf{x}(i)$ . The responses are  $\mathbf{y} = [\langle \mathbf{x}(1), \mathbf{w}^* \rangle, \dots, \langle \mathbf{x}(n), \mathbf{w}^* \rangle] \in \mathbb{R}^n$ . Define the input matrix  $\mathbf{Z}_0 \in \mathbb{R}^{(d+1)\times (n+1)}$  as:

$$\mathbf{Z}_0 = \begin{bmatrix} \mathbf{x}(1) & \mathbf{x}(2) & \cdots & \mathbf{x}(n) & \mathbf{x}(n+1) \\ \mathbf{y}(1) & \mathbf{y}(2) & \cdots & \mathbf{y}(n) & 0 \end{bmatrix}, \tag{2.1}$$

where the zero corresponds to the unknown response for  $\mathbf{x}(n+1)$ . The task is to predict  $(\mathbf{w}^*)^{\top}\mathbf{x}(n+1)$  using  $\mathbf{Z}_0$ . The training data consists of pairs  $(\mathbf{Z}_0, (\mathbf{w}^*)^{\top}\mathbf{x}(n+1))$  for  $\mathbf{x}(i) \sim \mathcal{D}_{\mathbf{X}}$  and  $\mathbf{w}^* \sim \mathcal{D}_{\mathbf{W}}$ .

**Self-Attention Without Softmax.** We focus on the linear self-attention layer, building on (Schlag et al., 2021; Von Oswald et al., 2023a). Let  $\mathbf{Z} \in \mathbb{R}^{(d+1)\times (n+1)}$  be the input matrix of n+1 tokens in  $\mathbb{R}^{d+1}$ . Standard self-attention layer is defined as

$$Attn_{smax}(\mathbf{Z}) := W_v \mathbf{Z} M \cdot smax(\mathbf{Z}^\top W_k^\top W_q \mathbf{Z}), \tag{2.2}$$

where  $W_v, W_k, W_q \in \mathbb{R}^{(d+1)\times (d+1)}$  are weight matrices, and  $smax(\cdot)$  denotes the column-wise softmax. The masking matrix M ensures that the label for  $\mathbf{x}(n+1)$  is excluded is given by

$$M = \begin{bmatrix} \mathbf{I}_n & 0 \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}.$$
 (2.3)

Omitting softmax, the attention mechanism becomes

$$Attn_{P,Q}(\mathbf{Z}) := P\mathbf{Z}M(\mathbf{Z}^{\top}Q\mathbf{Z}), \tag{2.4}$$

where  $P = W_v$  and  $Q = W_k^{\top} W_q$ . This simplified form, as shown in Ahn et al. (2024), can implement preconditioned gradient descent, and it is the one we also use.

**Architecture.** Following the related work, we also simplify the Transformer to consider only attention layers, using L layers of linear self-attention with a residual connection. Therefore, for each layer  $\ell$ , the output is updated as

$$\mathbf{Z}_{\ell+1} = \mathbf{Z}_{\ell} + \frac{1}{n} \operatorname{Attn}_{P_{\ell}, Q_{\ell}}(\mathbf{Z}_{\ell}), \quad \ell = 0, 1, \dots, L - 1.$$
 (2.5)

Using (2.5), with the input  $\mathbb{Z}_0$ , the final Transformer output is

$$TF_L(\mathbf{Z}_0; \{P_\ell, Q_\ell\}_{\ell=0}^{L-1}) = -[\mathbf{Z}_L]_{(d+1),(n+1)}.$$
(2.6)

The set of parameters  $\{P_{\ell}, Q_{\ell}\}_{\ell=0}^{L-1}$  is then learned by minimizing the following training objective:

$$\mathbb{E}_{(\mathbf{Z}_0, \mathbf{w}^*)} \left[ \left( \mathrm{TF}_L(\mathbf{Z}_0) + (\mathbf{w}^*)^\top \mathbf{x} (n+1) \right)^2 \right]. \tag{2.7}$$

We will utilize the following lemma from Ahn et al. (2024), which demonstrates that multi-layer Transformers simulate preconditioned gradient descent under suitable parameterization. We have provided the full proof of this lemma in Appendix A for completeness.

$$P_{\ell} = \begin{bmatrix} \mathbf{B}_{\ell} = 0_{d \times d} & 0 \\ 0 & 1 \end{bmatrix}, \quad Q_{\ell} = -\begin{bmatrix} \mathbf{A}_{\ell} & 0 \\ 0 & 0 \end{bmatrix}$$
 (2.8)

**Lemma 2.1** (Lemma 1, Ahn et al. (2024)). Let an L-layer linear transformer be parameterized by  $\mathbf{A}_0, \ldots, \mathbf{A}_{L-1}$ , as in (2.8). Let  $y_{\ell}^{(n+1)} = [\mathbf{Z}_{\ell}]_{(d+1),(n+1)}$  for  $\ell = 1, \ldots, L$ ; then,

$$y_{\ell}^{(n+1)} = -\langle \mathbf{x}^{(n+1)}, \mathbf{w}_{\ell}^{\text{gd}} \rangle, \tag{2.9}$$

where the sequence  $\{\mathbf{w}^{gd}_{\ell}\}$  is defined as  $\mathbf{w}^{gd}_{0}=0$  and for  $\ell=1,\ldots,L-1$ :

$$\mathbf{w}_{\ell+1}^{\mathrm{gd}} = \mathbf{w}_{\ell}^{\mathrm{gd}} - \mathbf{A}_{\ell} \nabla R_{\mathbf{w}^*}(\mathbf{w}_{\ell}^{\mathrm{gd}}), \tag{2.10}$$

with the empirical least-squares loss (with  $\mathbf{X} := [\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}] \in \mathbb{R}^{d \times n}$ ):

$$R_{\mathbf{w}^*}(\mathbf{w}) := \frac{1}{2n} \|\mathbf{X}^\top \mathbf{w} - \mathbf{X}^\top \mathbf{w}^*\|^2.$$
 (2.11)

*Note*. As observed in §C.1 of Ahn et al. (2024), the term  $Attn_{P_{\ell},Q_{\ell}}(\mathbf{Z}_{\ell})$  in (2.5) corresponds to the preconditioned gradient  $\mathbf{A}_{\ell}\nabla R_{\mathbf{w}^*}(\mathbf{w}^{\mathrm{gd}}_{\ell})$  of (2.11) in the update for  $\mathbf{w}^{\mathrm{gd}}_{\ell+1}$ .

### 2.2 Linear First-Order Methods (LFOMs)

Linear First-Order Methods (LFOMs) (Goh, 2017) optimize smooth functions by iteratively updating a parameter vector **w** with current and past gradients. The general update is

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \alpha_k \mathbf{d}^k, \tag{2.12}$$

where  $\alpha_k$  is a step size and  $\mathbf{d}^k$  is a direction typically tied to  $\nabla f(\mathbf{w}^k)$ . Different LFOMs vary in how  $\mathbf{d}^k$  and  $\alpha_k$  are chosen. For example:

(GD) 
$$\mathbf{w}^{k+1} = \mathbf{w}^0 - \alpha \sum_{i=0}^k \nabla f(\mathbf{w}^i),$$
  
(Momentum)  $\mathbf{w}^{k+1} = \mathbf{w}^0 + \sum_{i=0}^k \gamma_i^k \nabla f(\mathbf{w}^i).$ 

In more advanced LFOMs, the scalars  $\gamma_i^k$  are replaced by diagonal matrices  $\Gamma_i^k$ , enabling coordinatewise scaling.

Conjugate Gradient Descent (CGD). Specialized for quadratic minimization, CGD generates update directions  $s_n$  that are conjugate to previous ones, leading to faster convergence than standard gradient descent. It iterates:

$$\Delta \mathbf{w}_n = -\nabla f(\mathbf{w}_n), \quad \gamma_n = \frac{\|\nabla f(\mathbf{w}_n)\|^2}{\|\nabla f(\mathbf{w}_{n-1})\|^2},$$

$$\mathbf{s}_n = \Delta \mathbf{w}_n + \gamma_n \mathbf{s}_{n-1},$$
(2.13)

then chooses a step size  $\alpha_n$  (often via line search) and updates:  $\mathbf{w}_{n+1} = \mathbf{w}_n + \alpha_n \mathbf{s}_n$ .

Both momentum GD and CGD are LFOMs. Momentum methods are widely adopted in modern optimization, while CGD converges in at most *N* iterations for *N*-dimensional quadratics, and is effective for ill-conditioned problems.

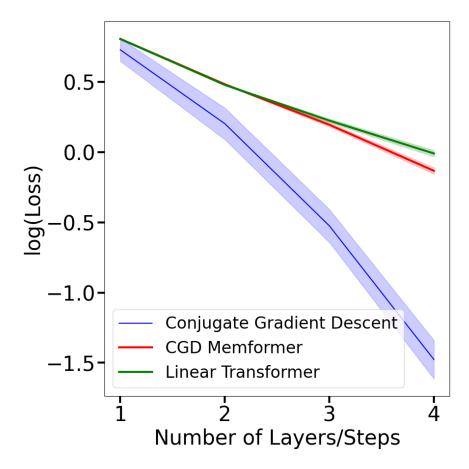


Figure 1: CGD-like Memformer (3.2) without preconditioning ( $A_{\ell} = I$ ) vs. actual CGD running separately on each test sample. Test data is drawn from the same distribution as the training data.

## 3 Memformers Can Implement LFOMs

Memformers can "learn" LFOMs in the sense described in Section 1. Each layer  $\ell$  has learnable parameters  $\mathbf{A}_{\ell}$ ,  $\mathbf{B}_{\ell}$  (2.8) and either  $\alpha_{\ell}$ ,  $\gamma_{\ell}$  (3.2) or  $\Gamma_{\ell}$  (3.4). Theorems 3.1 and 3.2 show that with suitable parameters, Memformers can *exactly implement* CGD and LFOM iterations in their forward pass.

With only a small number of learned parameters shared across a batch of in-context test data samples, Memformers can perform "CGD-like" (Section 3.1) or "LFOM-like" (Section 3.2) updates that rival or even surpass CGD. We refer to algorithms realized by (3.2) as "CGD-like" and those realized by (3.4) as "LFOM-like".

## 3.1 Single Dynamic Memory for CGD-like Algorithms

**Theorem 3.1.** A memory-augmented Transformer can implement Conjugate Gradient Descent (CGD) in its forward pass via a single memory register, where:

$$\mathbf{R}_{\ell} = \operatorname{Attn}_{P_{\ell}, Q_{\ell}}(\mathbf{Z}_{\ell}) + \gamma_{\ell} \mathbf{R}_{\ell-1}, \tag{3.1}$$

$$\mathbf{Z}_{\ell+1} = \mathbf{Z}_{\ell} + \alpha_{\ell} \frac{1}{n} \mathbf{R}_{\ell}, \tag{3.2}$$

and  $\gamma_{\ell}$ ,  $\alpha_{\ell}$  control the influence of past updates and step size, respectively.

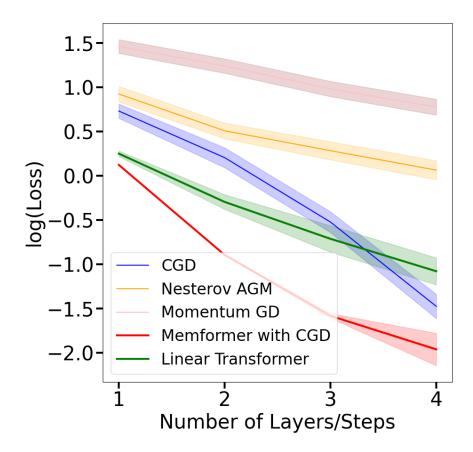


Figure 2: CGD-like Memformer (3.2) with preconditioning ( $A_{\ell} \neq I$ ). This yields a more general LFOM-like scheme, often outperforming CGD, Nesterov AGM and momentum GD. Test data is independently drawn from the same distribution as training data.

*Proof Sketch.* Here,  $\mathbf{R}_{\ell}$  tracks a *single* memory register across layers. CGD updates its search direction by combining the current gradient with the previous direction (2.13). The Transformer mimics this by combining  $\operatorname{Attn}_{P_{\ell},Q_{\ell}}(\mathbf{Z}_{\ell})$  (analogous to the gradient) and  $\gamma_{\ell}\mathbf{R}_{\ell-1}$ , and updating the next state  $\mathbf{Z}_{\ell+1}$  via (3.2). With  $\mathbf{A}_{\ell}=\mathbf{I}$ , this matches CGD applied to (2.11); see Appendix A for details.

## 3.2 Implementing k Steps of LFOM with Separate Memory Registers at Each Layer $\ell$

Next, we show how Transformers can simulate k steps of general LFOM updates (1.1) by maintaining a separate memory register at each layer  $\ell$ .

**Theorem 3.2.** A memory-augmented Transformer can implement k steps of LFOM in its forward pass by maintaining separate memory registers across layers  $\ell$ :

$$\mathbf{R}_{\ell} = \operatorname{Attn}_{P_{\ell}, Q_{\ell}}(\mathbf{Z}_{\ell}), \tag{3.3}$$

$$\mathbf{Z}_{\ell+1} = \mathbf{Z}_{\ell} + \frac{1}{n} \sum_{j=0}^{\ell} \Gamma_j^{\ell} \odot \mathbf{R}_{j}, \tag{3.4}$$

where  $\Gamma_i^{\ell}$  weights previous layer updates and  $\odot$  is a Hadamard product for scaling. <sup>1</sup>

<sup>&</sup>lt;sup>1</sup>The update (3.4) can be interpreted as a form of gated memory. Analogies with LSTMs or GRUs (which also use

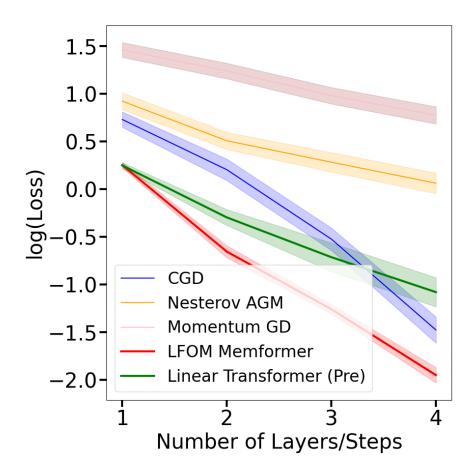


Figure 3: LFOM Memformer (3.4) vs. CGD, Nesterov AGM and momentum GD (Pre = non-trivial preconditioners).

*Proof Sketch.* Here,  $\mathbf{R}_{\ell}$  is a *separate* register at each layer. At layer  $\ell$ ,  $\mathbf{R}_{\ell}$  captures the current update  $\operatorname{Attn}_{P_{\ell},Q_{\ell}}(\mathbf{Z}_{\ell})$ . The final output  $\mathbf{Z}_{\ell+1}$  is updated by summing over registers  $\mathbf{R}_{j}$  scaled through  $\Gamma_{j}^{\ell} \in \mathbb{R}^{(n+1)\times(d+1)}$ . This resembles cumulative gradient steps of general LFOMs with diagonal preconditioners (1.1). A full proof is in Appendix A.

The Hadamard product  $\odot$  modulates the influence of  $\mathbf{R}_j$ , analogous to gradient preconditioning. This setup subsumes the case of diagonal preconditioners  $\Lambda_i^k$  acting on gradients  $\nabla R_{\mathbf{w}^*}(\mathbf{w}_i^{\mathrm{gd}})$ , which in the general form looks like:

$$\mathbf{w}_{k+1}^{\mathrm{gd}} = \mathbf{w}_0 + \sum_{i=0}^k \Lambda_i^k \nabla R_{\mathbf{w}^*}(\mathbf{w}_i^{\mathrm{gd}}). \tag{3.5}$$

The matrices  $\Gamma_j^\ell \in \mathbb{R}^{(d+1)\times (n+1)}$  and  $\Lambda_i^k \in \mathbb{R}^{d\times d}$  serve similar roles, but their dimensions differ. We expect this memory architecture to be able to perform richer algorithms than LFOMs, though a formal characterization of its full potential remains to be done.

*Remark* 3.3. Crucially, unlike classical methods such as CGD, which "adapt" by computing step sizes ( $\alpha$ ) and conjugacy coefficients ( $\gamma$ ) separately for each sample—even within the same batch—our Memformer uses a single set of learned parameters shared across *all* samples. For a test batch of size B, CGD thus effectively uses 2B parameters per iteration, whereas the Memformer

Hadamard gating) suggest potential ways to refine memory usage in Transformers.

has just the trained internal parameters  $\{P_{\ell}, Q_{\ell}, \Gamma_{\ell}\}$  (3.4). This amounts to a meta-optimizer that can exploit "average-case" behavior over the data distribution, rather than re-optimizing each instance individually.

## 3.3 Experimental Results

We evaluate Memformers for learning CGD, general LFOMs, and LFOMs with GD<sup>++</sup> (a quasi-Newton method approximating the Hessian inverse via a truncated Neumann series (Von Oswald et al., 2023a).

**Setup.** We use the in-context loss (2.11) for linear regression with d=5 and n=20. Each run samples a random orthogonal matrix  $\mathbf{U} \in \mathbb{R}^{d \times d}$  and uses  $\mathbf{D} = \mathrm{diag}(1,1,\frac{1}{2},\frac{1}{4},1)$ . With a variance scalar  $\sigma^2$  (default 1), we set  $\mathbf{\Sigma} = \sigma^2 \mathbf{U}^{\mathsf{T}} \mathbf{D} \mathbf{U}$ . We draw inputs  $\mathbf{x}^{(i)}$  i.i.d. from  $\mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$ , and targets  $\mathbf{w}^*$  from  $\mathcal{N}(\mathbf{0}, \mathbf{\Sigma}^{-1})$ . We optimize the in-context prediction function f (2.7) using a 4-layer linear Transformer trained via ADAM, initializing parameters  $\mathbf{A}_0$ ,  $\mathbf{A}_1$ ,  $\mathbf{A}_2$ ,  $\mathbf{A}_3$  (2.8) i.i.d. Gaussian, with batch size 1000 (resampled every 100 steps) and gradient clipping of 0.01. Plots average over five independent runs with different  $\mathbf{U}$ .

**CGD-like Architecture.** Figure 1 shows a Memformer parameterized as in (3.2), where the globally learned  $\alpha_{\ell}$  and  $\gamma_{\ell}$  emulate CGD's line search and deflection. Unlike standard CGD, which refines these parameters individually for each sample, our Memformer uses a single set of learned scalars shared across all data. Despite this approximation, performance remains competitive. Figure 2 extends the approach by allowing non-scalar preconditioners  $A_{\ell}$  (2.8), thus making the method a more general "LFOM-like" strategy that can even outperform standard CGD.

**LFOM Memformer.** Figure 3 shows a Memformer with the LFOM-like architecture (3.4), where each layer's  $\Gamma_{\ell}$  acts as a preconditioner. In our experiments, we consider the special case of  $\Gamma_{j}^{\ell} = \Gamma_{j} \ \forall \ell$ , which is more natural, if we consider that each layer j of the Memformer has an associated  $\Gamma_{j}$ . In practice, these parameters further improve performance. Figure 4 extends LFOM with GD<sup>++</sup> by allowing the  $\mathbf{B}_{\ell}$  blocks (2.8) to be non-zero. In this case, the  $\mathbf{B}_{\ell}$  matrices resemble a heavily truncated Neumann series of the inverse  $\mathbf{X}\mathbf{X}^{\top}$  (Hessian of (2.11)). Our experiments with more than 4 layers are detailed in Appendix C.

#### 3.4 Influence of Batch Size on Performance

The results in Section 3.3 compares the performance of Transformers and Memformers (which learn shared generic parameters during training) against CGD, which computes specific parameters for each observation in a batch of size B=1000, independently resampled from the same distribution. While CGD optimizes each observation individually, Transformers and Memformers use shared parameters  $P_\ell$ ,  $Q_\ell$  (and  $\alpha_\ell$ ,  $\gamma_\ell$ , or  $\Gamma_\ell$ ) for each layer  $\ell$ , uniformly used for all 1000 observations in the batch. The average log-loss versus layers are plotted for comparison in Figures 2 and 3.

The strength of LFOM Memformers (3.4) (with matrices  $\Gamma_{\ell}$  restricted to scalar multiples of the identity) **becomes even more pronounced when trained on smaller batch sizes, such as** B=1 **and** B=10. In these cases, Memformers learn parameters that significantly outperform CGD, even when CGD runs independently on each observation in the batch. Figures 5 and 6 illustrate this comparison.

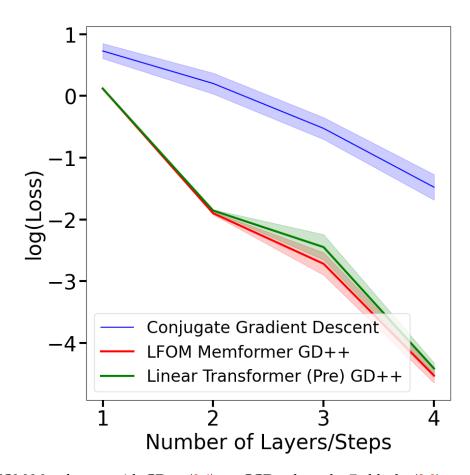


Figure 4: LFOM Memformer with GD++ (3.4) vs. CGD, where the  $\mathbf{B}_{\ell}$  blocks (2.8) approximate the Hessian inverse (quasi-Newton).

## 4 Multi-Headed Attention and Out-of-Distribution (OOD) Performance

### 4.1 OOD Adaptation using Multi-Headed Attention

Next, we explore how multi-head attention further improves both in-distribution and OOD performance. In our experiments, increasing the number of heads yields notable gains in test-loss metrics. By learning diverse preconditioning matrices, multi-head architectures help Memformers adapt to data with varying covariance structures. Concretely, in the update (3.1), we sum the attention outputs from different heads into a single memory register  $\mathbf{R}_\ell$  at each layer, thus aggregating multiple "views" of the underlying data distribution. These findings align with prior studies underscoring the critical role of multi-head attention in in-context learning. For example, Chen et al. (2024) highlight its importance in sparse linear regression, and Cui et al. (2024) demonstrate its theoretical and empirical benefits. As illustrated in Figure 7, the number of heads strongly influences both convergence speed and final test loss.

## 4.2 Test-Time Adaptation and Gated Memory

Building on the above, we next consider test-time adaptation and show how gating mechanisms allow robust performance under distribution shift.

Mixture of Experts (MoE) for Variance Adaptation. To extend multi-head architectures

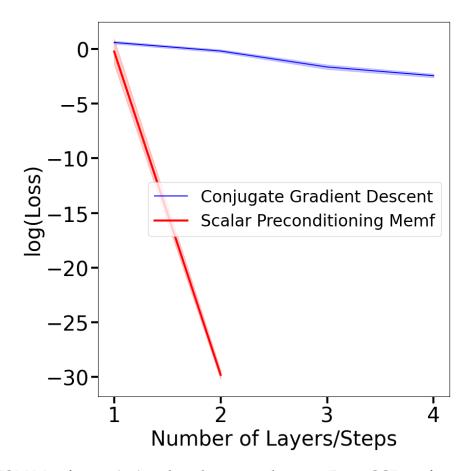


Figure 5: LFOM Memformer (3.4) with scalar preconditioners  $\Gamma_{\ell}$  vs. CGD performance on small batch size (B=1). The Memformer demonstrates superior performance.

to OOD scenarios, we employ a *mixture of experts (MoE)* approach Jacobs et al. (1991) in a 3-head, 4-layer Memformer. Each head specializes in Gaussian inputs with different variances  $\sigma^2 \in \{1.0, 2.0, 3.0\}$ . At inference, a learnable gating mechanism scales the layerwise output  $\sum_{j=0}^{\ell} \Gamma_j \odot \mathbf{R}_j$  (3.4) per head coordinate-wise. Concretely, each head  $H_i$  has a scalar gating coefficient  $\alpha_i \in \mathbb{R}$ , enabling the model to adapt effectively to unseen variances (e.g.,  $\sigma^2 = \{0.5, 1.5, 3.5\}$ ). As shown in Figure 8, this design ensures robust generalization even when the variance differs from the training set.

Gaussian Mixture Models (GMMs). We further examine OOD adaptation in a 3-head, 4-layer Memformer tested on a Gaussian Mixture Model (GMM) with different means ( $\{2.5, 0.0, -1.0\}$ ), variances ( $\{0.5, 1.0, 2.0\}$ ), and mixture weights ( $\{0.2, 0.3, 0.5\}$ ). Each head specializes in one mixture component. As before, a gating mechanism similarly scales the layerwise output  $\sum_{j=0}^{\ell} \Gamma_j \odot \mathbf{R}_j$  across heads.

For this GMM setup, we observe that using a 3-head, 4-layer Linear Transformer (cf. Figure 9) gives better performance than a 3-head, 4-layer Memformer (cf. Figure 10), as the  $\Gamma_j$  matrices of each head in the Memformer overfit to the individual GMM components. However, this issue can be overcome, with a higher test-time budget, if the  $\Gamma_j$  matrices are also allowed to be trainable at test time (see Figure 10).

**Theorem 4.1** (Multi-Head Memformer with Soft Gating). Consider a multi-head Memformer with H heads, each parameterized by  $\{P_{\ell}^h, Q_{\ell}^h, \Gamma_{\ell}^h\}$ . Suppose  $\mathbf{Z}_0$  is drawn from a mixture of M Gaussian components

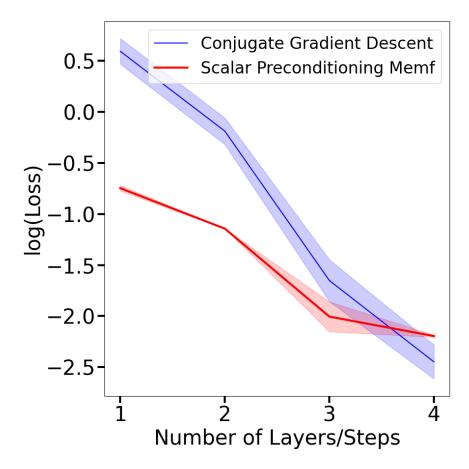


Figure 6: LFOM Memformer (3.4) with scalar preconditioners  $\Gamma_{\ell}$  vs. CGD performance on small batch size (B=10). The Memformer outperforms CGD on the training data.

 $\{\mathcal{N}(\mu_m, \Sigma_m)\}_{m=1}^M$ , where each  $\mathbf{x}^{(i)} \sim \mathcal{N}(\mu_m, \Sigma_m)$  and  $\mathbf{w}^* \sim \mathcal{N}(\mu_m, \Sigma_m^{-1})$ . If each component m has at least one head  $h_m$  achieving near-optimal first-order performance, then there exist scalar gating coefficients  $\alpha_{m,h}$  such that for any new prompt from component m, the Memformer achieves near-optimal performance v is soft-gating:

$$\alpha_{m,1} \operatorname{Head}_1 + \cdots + \alpha_{m,H} \operatorname{Head}_H$$

In particular, if  $\alpha_{m,h_m}=1$  and  $\alpha_{m,h}=0$  for  $h\neq h_m$ , the model's updates align exactly with head  $h_m$ 's near-optimal method for  $\mathcal{N}(\mu_m,\Sigma_m)$ .

A formal proof of Theorem 4.1 appears in Appendix A. As indicated above, a Transformer can handle OOD data by fine-tuning specific test-time parameters (e.g., the gating coefficients or the diagonal of  $\Gamma_i$ , as in Figure 10). This design in effect operates as a meta-version of incontext learning: the main architecture parameters remain fixed, while gating and memory preconditioners can adjust to accommodate distribution shifts, depending on the test-time budget.

<sup>&</sup>lt;sup>2</sup>If  $\sum_{h=1}^{H} \alpha_{m,h} = 1$ , then  $\alpha_{m,1}$  Head<sub>1</sub> + · · · +  $\alpha_{m,H}$  Head<sub>H</sub> is a convex combination of the H heads. Such linear updates also appear in momentum/AdamW, where  $(1 - \eta \lambda)\theta_t + \eta \lambda$  (gradient-based step) is itself a convex combination.

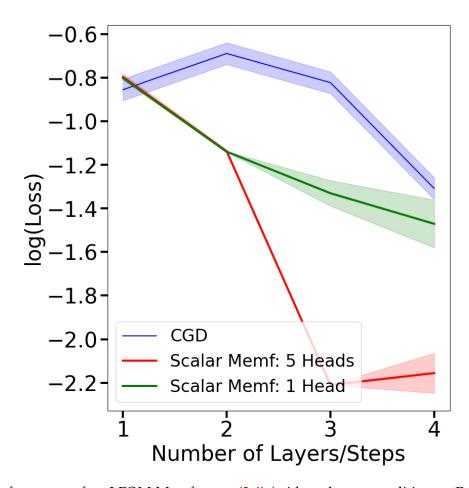


Figure 7: Performance of an LFOM Memformer (3.4) (with scalar preconditioners  $\Gamma_{\ell}$ ) comparing one-head and five-head attention, relative to CGD. Multi-head attention accelerates convergence and improves test performance.

## 5 LFOMs as Learnable Algorithms

Finally, LFOM updates can themselves be *learned* as a global meta-optimizer, bridging the gap between classical algorithms and in-context algorithmic learning. One can directly learn the parameters of a linear first-order method (LFOM) from data, treating it as a global algorithm that generalizes across random in-context samples. The following theorem provides a standard finite-sample guarantee for such an approach under mild regularity conditions (boundedness, sub-Gaussianity, continuity, and compactness). A complete version and proof appear in Appendix A.

**Theorem 5.1** (Statistical Learnability of LFOMs in the In-Context Setting). *Fix a class of LFOMs parameterized by*  $\theta \in \Theta \subset \mathbb{R}^p$ , *e.g.*, *via diagonal preconditioners. Each LFOM*  $\theta$  *maps an in-context prompt*  $\mathbf{Z}_0$  *to a final prediction*  $\hat{y}_{\theta}(\mathbf{Z}_0)$  *for*  $\mathbf{x}(n+1)$ , *incurring the loss*  $\ell(\theta; \mathbf{Z}_0) = (\hat{y}_{\theta}(\mathbf{Z}_0) - \langle \mathbf{x}(n+1), \mathbf{w}^* \rangle)^2$ .

Drawing N i.i.d. prompts  $\{\mathbf{Z}_0^{(i)}\}_{i=1}^N$  from  $\mathcal{D}$ , define  $\widehat{\theta} = \arg\min_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^N \ell(\theta; \mathbf{Z}_0^{(i)})$ . Then, with high probability over the sample draw,

$$\mathbb{E}_{\mathbf{Z}_0 \sim \mathcal{D}} \big[ \ell(\widehat{\theta}; \mathbf{Z}_0) \big] \leq \min_{\theta \in \Theta} \mathbb{E}_{\mathbf{Z}_0 \sim \mathcal{D}} \big[ \ell(\theta; \mathbf{Z}_0) \big] + \epsilon(N, \delta),$$

where  $\epsilon(N, \delta) \to 0$  as  $N \to \infty$ . In other words, the learned LFOM parameters  $\widehat{\theta}$  generalize well on new in-context samples from  $\mathcal{D}$ .

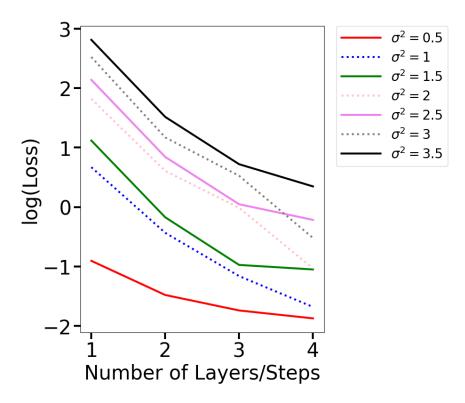


Figure 8: MoE-based 3-head, 4-layer Memformer (3.4) evaluated on Gaussian inputs of varying unseen variances. A trainable gating mechanism allows effective OOD adaptation.

Note that our Memformer update rule (3.4) encompasses an even broader family of optimization strategies than classical LFOMs. Figure 11 illustrates how learning LFOM parameters (e.g. diagonal preconditioners) directly from data (using ADAM or similar) can match or exceed the performance of Conjugate Gradient Descent (CGD), while using one shared set of parameters across multiple samples. Such a strategy can overfit if batches are extremely small, but tends to generalize effectively for moderate to large batch sizes.

## 6 Conclusion and Future Work

We find that memory-augmented Transformers (Memformers) can learn and implement a wide range of first-order optimization algorithms, including gradient descent, conjugate gradient, and momentum methods. This versatility highlights their potential as general-purpose metaoptimizers. Below are several avenues for further exploration:

Architectural Extensions, MoE, and TTA. Additional gating strategies, mixture-of-experts (MoE), and refined memory modules may boost performance on out-of-distribution (OOD) data. Even a small number of experts can deliver robust gains, though identifying the "right" number of experts remains open. More advanced test-time adaptation (TTA), such as fine-tuning the gating or expert parameters on new data, could further enhance generalization to shifting distributions.

**Beyond Quadratic Objectives.** Most theoretical work on algorithm learning in Transformers has focused on linear regression, leaving open the understanding of more complex, potentially nonconvex objectives. By integrating nonlinear attention layers or MLP components, Memformers could tackle tasks like logistic regression or PDE-based inverse problems, where simple quadratic

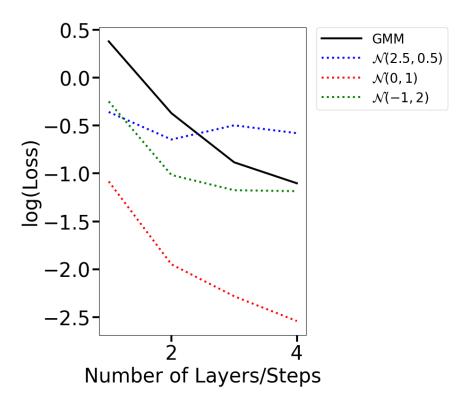


Figure 9: A 3-head, 4-layer Linear Transformer tested on a GMM. Each head specializes in one mixture component, and a gating module adjusts the model for OOD data.

assumptions are insufficient. Analyzing Memformers' in-context learning on such domains could unify meta-learning research with more realistic, large-scale optimization problems, while clarifying theoretical guarantees beyond the linear-quadratic case.

**Efficiency vs. Generalization.** While attention-based models can be more computationally demanding than methods like momentum or CGD, they exhibit strong generalization across diverse tasks. Systematic studies of runtime and accuracy trade-offs could clarify the conditions under which learned optimizers outperform classical approaches.

**Meta-Learning and Transfer.** Memformers naturally connect to meta-learning by reusing learned optimization strategies. Future investigations might examine more sophisticated TTA, gating, or lightweight fine-tuning approaches to handle distribution shifts. Understanding how best to structure MoE—such as how many experts to deploy and how to select them on the fly—promises both practical efficiency and robust transfer in a variety of domains.

Limitations of our framework. While Memformers are versatile, our experiments (Figures 2, 3) show they do not drastically outperform advanced methods on quadratic tasks (2.11), likely due to the task structure. Future work on broader ICL formulations may shed light on this matter. Notably, Transformers can implement second-order methods like Newton's (Fu et al., 2023; Giannou et al., 2024), which typically converge faster and more accurately than LFOMs. However, our focus in this paper is on exploring first-order optimization algorithms that augmented Transformers can learn, rather than proposing a one-size-fits-all optimizer.

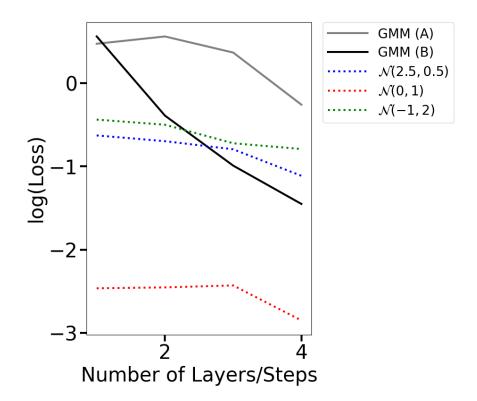


Figure 10: A 3-head, 4-layer Memformer (3.4) tested on a GMM. The gray line shows results with a learned gating scheme, while the black line also includes test-time updates to the  $\Gamma_i$  matrices.

## Reproducibility Statement

We believe the following points provide a clear path for replicating our results:

- Code Availability: The code for our experiments, including Memformers and LFOM implementations, is available at https://anonymous.4open.science/r/ICML-2025-LFOM\_Memformer.
- Experiment Setup: Detailed descriptions of the training setup, model architecture, parameter initialization, and optimization methods are included in Sections 2 and 3.3.
- **Random Seeds**: Random seeds were fixed across all experiments to ensure consistency, and they are provided in the code repository for replication.
- **Hardware Requirements**: All experiments were conducted on NVIDIA T4 GPUs in Google Colab.

## References

Tom B Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*, 2021.

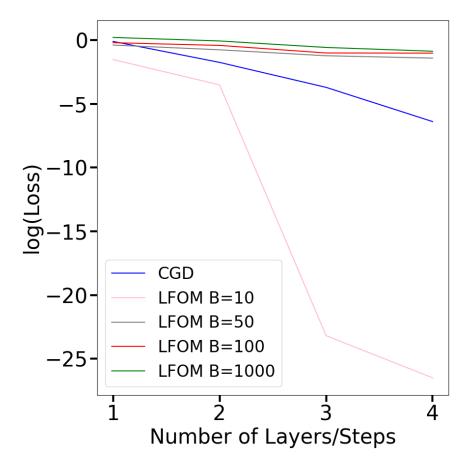


Figure 11: Comparison of a learned LFOM approach vs. standard Conjugate Gradient Descent (CGD) on a synthetic quadratic problem (2.11). LFOM parameters (3.5) are trained directly from data, while CGD is run separately on each instance. (B = Batch Size)

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv* preprint arXiv:2104.08786, 2021.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Zhiyong Wu, Yaoxiang Wang, Jiacheng Ye, and Lingpeng Kong. Self-adaptive in-context learning: An information compression perspective for in-context example selection and ordering. *arXiv* preprint arXiv:2212.10375, 2022.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.

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

Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Shuming Ma, Zhifang Sui, and Furu Wei. Why can gpt learn in-context? language models implicitly perform gradient descent as meta-optimizers.

- arXiv preprint arXiv:2212.10559, 2022.
- Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pages 35151–35174. PMLR, 2023a.
- Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598, 2022.
- Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*, 2022.
- Arvind Mahankali, Tatsunori B Hashimoto, and Tengyu Ma. One step of gradient descent is provably the optimal in-context learner with one layer of linear self-attention. *arXiv* preprint *arXiv*:2307.03576, 2023.
- Kwangjun Ahn, Xiang Cheng, Hadi Daneshmand, and Suvrit Sra. Transformers learn to implement preconditioned gradient descent for in-context learning. *Advances in Neural Information Processing Systems*, 36, 2024.
- Deqing Fu, Tian-Qi Chen, Robin Jia, and Vatsal Sharan. Transformers learn higher-order optimization methods for in-context learning: A study with linear models. *arXiv preprint arXiv:2310.17086*, 2023.
- Max Vladymyrov, Johannes Von Oswald, Mark Sandler, and Rong Ge. Linear transformers are versatile in-context learners. *arXiv preprint arXiv:2402.14180*, 2024.
- Gabriel Goh. Why momentum really works. *Distill*, 2017. doi: 10.23915/distill.00006. URL http://distill.pub/2017/momentum.
- Qingyang Wu, Zhenzhong Lan, Kun Qian, Jing Gu, Alborz Geramifard, and Zhou Yu. Memformer: A memory-augmented transformer for sequence modeling. *arXiv preprint arXiv:2010.06891*, 2020.
- Dongkuan Xu, Junjie Liang, Wei Cheng, Hua Wei, Haifeng Chen, and Xiang Zhang. Transformer-style relational reasoning with dynamic memory updating for temporal network modeling. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 4546–4554, 2021.
- Ruiqi Zhang, Spencer Frei, and Peter L Bartlett. Trained transformers learn linear models incontext. *Journal of Machine Learning Research*, 25(49):1–55, 2024.
- Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International Conference on Machine Learning*, pages 9355–9366. PMLR, 2021.
- Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *ICLR* 2025 arXiv:2408.03314, 2024.
- Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *International conference on machine learning*, pages 9229–9248. PMLR, 2020.
- Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. *arXiv preprint arXiv:2111.02080*, 2021.
- Johannes Von Oswald, Eyvind Niklasson, Maximilian Schlegel, Seijin Kobayashi, Nicolas Zucchet, Nino Scherrer, Nolan Miller, Mark Sandler, Max Vladymyrov, Razvan Pascanu, et al. Uncovering

- mesa-optimization algorithms in transformers. arXiv preprint arXiv:2309.05858, 2023b.
- Michael Hahn and Navin Goyal. A theory of emergent in-context learning as implicit structure induction. *arXiv preprint arXiv:2303.07971*, 2023.
- Kwangjun Ahn, Xiang Cheng, Minhak Song, Chulhee Yun, Ali Jadbabaie, and Suvrit Sra. Linear attention is (maybe) all you need (to understand transformer optimization). *arXiv preprint arXiv:2310.01082*, 2023.
- Angeliki Giannou, Liu Yang, Tianhao Wang, Dimitris Papailiopoulos, and Jason D Lee. How well can transformers emulate in-context newton's method? *arXiv preprint arXiv*:2403.03183, 2024.
- Xingwu Chen, Lei Zhao, and Difan Zou. How transformers utilize multi-head attention in in-context learning? a case study on sparse linear regression. *arXiv preprint arXiv:2408.04532*, 2024.
- Yingqian Cui, Jie Ren, Pengfei He, Jiliang Tang, and Yue Xing. Superiority of multi-head attention in in-context linear regression. *arXiv preprint arXiv:2401.17426*, 2024.
- Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.
- Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.
- Sara A Geer. Empirical Processes in M-estimation, volume 6. Cambridge university press, 2000.

## **Supplementary Material**

## A Proofs

## A.1 Proof of Lemma 2.1: Equivalence to Preconditioned Gradient Descent

This proof already exists in the literature, for instance, in Subsection C.1 of Ahn et al. (2024). However, we repeat it here, to make this paper as self-contained as possible.

Consider a set of fixed samples  $\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(n)}$ , along with a fixed vector  $\mathbf{w}^*$ . Let  $P=\{P_i\}_{i=0}^k$  and  $Q=\{Q_i\}_{i=0}^k$  represent fixed weights, and let  $\mathbf{Z}_i$  evolve as per equation (2.5). Define  $\mathbf{X}_i$  as the first d rows of  $\mathbf{Z}_k$  (under equation (2.8), we have  $\mathbf{X}_i=\mathbf{X}_0$  for all i), and let  $\mathbf{Y}_i$  be the (d+1)-th row of  $\mathbf{Z}_i$ . Now, let  $g(\mathbf{x},\mathbf{y},k):\mathbb{R}^d\times\mathbb{R}\times\mathbb{Z}\to\mathbb{R}$  be a function such that for  $\mathbf{x}_{n+1}=\mathbf{x}$  and  $\mathbf{y}_{n+1}^{(0)}=\mathbf{y}$ , the function is defined as  $g(\mathbf{x},\mathbf{y},k):=\mathbf{y}_{n+1}^{(k)}$ . It's worth noting that  $\mathbf{y}_{n+1}^{(k)}=[\mathbf{Y}_k]_{n+1}$ .

We can verify that, under equation (2.8), the update rule for  $\mathbf{y}_{n+1}^{(k)}$  is given by:

$$\mathbf{Y}_{k+1} = \mathbf{Y}_k - \frac{1}{n} \mathbf{Y}_k M \mathbf{X}_0^{\top} A_k \mathbf{X}_0, \tag{A.1}$$

where *M* is a mask matrix of the form:

$$M = \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix}.$$

The following points can be verified:

1.  $g(\mathbf{x}, \mathbf{y}, k) = g(\mathbf{x}, 0, k) + \mathbf{y}$ . To see this, note that for each  $i \in \{1, ..., n\}$ , we have:

$$\mathbf{y}_{k+1}^{(i)} = \mathbf{y}_k^{(i)} - \frac{1}{n} \sum_{i=1}^n \mathbf{x}^{(i)\top} A_k \mathbf{x}^{(j)} \mathbf{y}_k^{(j)}.$$

Thus,  $\mathbf{y}_k^{(i)}$  does not depend on  $\mathbf{y}_{n+1}^{(t)}$  for any t. For  $\mathbf{y}_{n+1}^{(k)}$ , the update becomes:

$$\mathbf{y}_{n+1}^{(k+1)} = \mathbf{y}_{n+1}^{(k)} - \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_{n+1}^{\top} A_k \mathbf{x}^{(j)} \mathbf{y}_k^{(j)},$$

which clearly shows that the dependence on  $\mathbf{y}_{n+1}^{(k)}$  is additive. Through a simple induction, we can establish:

$$g(\mathbf{x}, \mathbf{y}, k+1) - \mathbf{y} = g(\mathbf{x}, \mathbf{y}, k) - \mathbf{y}.$$

2. The function  $g(\mathbf{x}, 0, k)$  is linear in  $\mathbf{x}$ . To see this, note that for  $j \neq n+1$ ,  $\mathbf{y}_j^{(k)}$  does not depend on  $\mathbf{x}_{n+1}^{(t)}$  for any t, j, or k. Therefore, the update for  $\mathbf{y}_{n+1}^{(k+1)}$  depends linearly on  $\mathbf{x}_{n+1}$  and  $\mathbf{y}_{n+1}^{(k)}$ . Since  $\mathbf{y}_{n+1}^{(0)} = 0$  is linear in  $\mathbf{x}$ , we conclude by induction that the result holds.

Considering these points, we can confirm that for each k, there exists a vector  $\theta_k \in \mathbb{R}^d$  such that:

$$g(\mathbf{x}, \mathbf{y}, k) = g(\mathbf{x}, 0, k) + \mathbf{y} = \langle \theta_k, \mathbf{x} \rangle + \mathbf{y},$$

for all  $\mathbf{x}$  and  $\mathbf{y}$ . It follows that  $g(\mathbf{x}, \mathbf{y}, 0) = \mathbf{y}$ , so that  $\langle \theta_0, \mathbf{x} \rangle = g(\mathbf{x}, \mathbf{y}, 0) - \mathbf{y} = 0$ , implying  $\theta_0 = 0$ . We now focus on the third key fact: for each i, we have:

$$g(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, k) = \mathbf{y}_k^{(i)} = \langle \theta_k, \mathbf{x}^{(i)} \rangle + \mathbf{y}^{(i)}.$$

To prove this, let  $\mathbf{x}_{n+1} := \mathbf{x}^{(i)}$  for some  $i \in \{1, ..., n\}$ . Then:

$$\mathbf{y}_{k+1}^{(i)} = \mathbf{y}_k^{(i)} - \frac{1}{n} \sum_{j=1}^n \mathbf{x}^{(i)\top} A_k \mathbf{x}^{(j)} \mathbf{y}_k^{(j)},$$

$$\mathbf{y}_{n+1}^{(k+1)} = \mathbf{y}_{n+1}^{(k)} - \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_{n+1}^{\top} A_k \mathbf{x}^{(j)} \mathbf{y}_k^{(j)},$$

therefore,  $\mathbf{y}_{k+1}^{(i)} = \mathbf{y}_{n+1}^{(k+1)}$  when  $\mathbf{y}_k^{(i)} = \mathbf{y}_{n+1}^{(k)}$ . This completes the induction, given that  $\mathbf{y}_0^{(i)} = \mathbf{y}_{n+1}^{(0)}$  by definition.

Let  $\bar{\mathbf{X}} \in \mathbb{R}^{d \times n}$  be the matrix whose columns are  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}$ , excluding  $\mathbf{x}_{n+1}$ , and let  $\bar{\mathbf{Y}}_k \in \mathbb{R}^{1 \times n}$  be the vector of  $\mathbf{y}_k^{(1)}, \dots, \mathbf{y}_k^{(n)}$ . It follows that:

$$\mathbf{\bar{Y}}_k = \mathbf{\bar{Y}}_0 + \mathbf{\theta}_k^{\top} \mathbf{\bar{X}}.$$

Using this, the update formula for  $\mathbf{y}_{n+1}^{(k)}$  becomes:

$$\mathbf{y}_{n+1}^{(k+1)} = \mathbf{y}_{n+1}^{(k)} - \frac{1}{n} \langle A_k \bar{\mathbf{X}}^\top \bar{\mathbf{Y}}_k, \mathbf{x}_{n+1} \rangle, \tag{A.2}$$

leading to the update:

$$\langle \theta_{k+1}, \mathbf{x}_{n+1} \rangle = \langle \theta_k, \mathbf{x}_{n+1} \rangle - \frac{1}{n} \langle A_k \bar{\mathbf{X}} (\bar{\mathbf{X}}^\top \theta_k + \bar{\mathbf{Y}}_0), \mathbf{x}_{n+1} \rangle. \tag{A.3}$$

Since  $x_{n+1}$  is arbitrary, we derive the general update formula:

$$\theta_{k+1} = \theta_k - \frac{1}{n} A_k \bar{\mathbf{X}} \bar{\mathbf{X}}^\top (\theta_k + \mathbf{w}^*). \tag{A.4}$$

Treating  $A_k$  as a preconditioner, and letting  $f(\theta) := \frac{1}{2n} (\theta + \mathbf{w}^*)^\top \bar{\mathbf{X}} \bar{\mathbf{X}}^\top (\theta + \mathbf{w}^*)$ , we can express the update as:

$$\theta_{k+1} = \theta_k - \frac{1}{n} A_k \nabla f(\theta). \tag{A.5}$$

Finally, let  $\mathbf{w}_k^{\text{gd}} := -\theta_k$ . We can verify that  $f(-\mathbf{w}) = R_{\mathbf{w}^*}(\mathbf{w})$ , implying that:

$$\mathbf{w}_{k+1}^{\mathrm{gd}} = \mathbf{w}_{k}^{\mathrm{gd}} - \frac{1}{n} A_{k} \nabla R_{\mathbf{w}^{*}}(\mathbf{w}_{k}^{\mathrm{gd}}). \tag{A.6}$$

We also confirm that for any  $\mathbf{x}_{n+1}$ , the prediction of  $\mathbf{y}_{n+1}^{(k)}$  is:

$$g(\mathbf{x}_{n+1},\mathbf{y}_{n+1},k)=\mathbf{y}_{n+1}-\langle\theta,\mathbf{x}_{n+1}\rangle=\mathbf{y}_{n+1}+\langle\mathbf{w}_k^{\mathrm{gd}},\mathbf{x}_{n+1}\rangle.$$

This concludes the proof. We have simply followed the update rule (2.5) to its logical conclusion.

#### A.2 Full Proof of Theorem 3.1

**Theorem.** A memory-augmented Transformer can implement Conjugate Gradient Descent (CGD) through a dynamic memory mechanism that recursively refines search directions, where the update rules are:

$$\mathbf{R}_{\ell} = \operatorname{Attn}_{P_{\ell}, Q_{\ell}}(\mathbf{Z}_{\ell}) + \gamma_{\ell} \mathbf{R}_{\ell-1}, \tag{A.7}$$

$$\mathbf{Z}_{\ell+1} = \mathbf{Z}_{\ell} + \alpha_{\ell} \frac{1}{n} \mathbf{R}_{\ell},\tag{A.8}$$

where  $\gamma_{\ell}$  and  $\alpha_{\ell}$  control past update influence and step size.

#### **Proof**

Our goal is to demonstrate that, under appropriate parameter configurations, the memory-augmented Transformer updates given by equations (A.7) and (A.8) correspond precisely to the Conjugate Gradient Descent (CGD) algorithm when applied to the quadratic loss function:

$$R_{\mathbf{w}^*}(\mathbf{w}) = \frac{1}{2n} (\mathbf{w} - \mathbf{w}^*)^{\top} \mathbf{X} \mathbf{X}^{\top} (\mathbf{w} - \mathbf{w}^*).$$
 (A.9)

We will establish a mapping between the Transformer's operations and the steps of the CGD algorithm, demonstrating that the Transformer can implement CGD under certain parameter settings.

#### **CGD** Algorithm for Quadratic Functions

For minimizing a quadratic function, the CGD algorithm proceeds as follows:

## Algorithm: Conjugate Gradient Descent (CGD)

- Initialize:  $\mathbf{w}_0$ ,  $\mathbf{r}_0 = -\nabla f(\mathbf{w}_0)$ ,  $\mathbf{s}_0 = \mathbf{r}_0$
- For n = 1, 2, ...:
  - Compute the residual:

$$\mathbf{r}_n = -\nabla f(\mathbf{w}_n)$$

- Compute the conjugacy coefficient:

$$\gamma_n = rac{\mathbf{r}_n^ op \mathbf{r}_n}{\mathbf{r}_{n-1}^ op \mathbf{r}_{n-1}}$$

- Update the search direction:

$$\mathbf{s}_n = \mathbf{r}_n + \gamma_n \mathbf{s}_{n-1}$$

- Compute the step size:

$$\alpha_n = \frac{\mathbf{r}_n^{\top} \mathbf{r}_n}{\mathbf{s}_n^{\top} \mathbf{H} \mathbf{s}_n}$$

- Update the parameters:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \alpha_n \mathbf{s}_n$$

EndFor

### **Mapping CGD Updates to Transformer Updates**

We first recall that in the proof of Lemma 1 (A.1), the  $\mathbf{w}_{k+1}^{\mathrm{gd}}$  update rule

$$\mathbf{w}_{k+1}^{\text{gd}} = \mathbf{w}_k^{\text{gd}} - \frac{1}{n} A_k \nabla R_{\mathbf{w}^*}(\mathbf{w}_k^{\text{gd}}), \tag{A.10}$$

is a direct downstream consequence of the  $\mathbb{Z}_{\ell+1}$  update rule (2.5)

$$\mathbf{Z}_{\ell+1} = \mathbf{Z}_{\ell} + \frac{1}{n} \operatorname{Attn}_{P_{\ell}, Q_{\ell}}(\mathbf{Z}_{\ell}), \quad \ell = 0, 1, \dots, L-1,$$

under the parameterization given in equation (2.8). Thus, the  $\operatorname{Attn}_{P_\ell,Q_\ell}$  term in the  $\mathbf{Z}_\ell$  update equation is, in a precise sense, paralleled by the  $-\frac{1}{n}A_k\nabla R_{\mathbf{w}^*}(\mathbf{w}_k^{\mathrm{gd}})$  term in the  $\mathbf{w}_{k+1}^{\mathrm{gd}}$  update equation (A.10).

#### Step 1: Initialization

• CGD:

$$\mathbf{w}_0$$
 given,  $\mathbf{r}_0 = -\nabla f(\mathbf{w}_0)$ ,  $\mathbf{s}_0 = \mathbf{r}_0$ .

- Transformer:
  - The initial state  $\mathbf{Z}_0$  in (2.5) parallels  $\mathbf{w}_0$  in (A.10).
  - The memory register **R** is initialized to  $Attn_{P_0,Q_0}(\mathbf{Z}_0)$ , i.e.,  $\mathbf{R}_0 = Attn_{P_0,Q_0}(\mathbf{Z}_0)$ , corresponding to  $\mathbf{s}_0 = \mathbf{r}_0$ .
  - We set  $\gamma_0 = 0$ , consistent with CGD initialization.

#### Step 2: Update Memory Register (Search Direction)

• Transformer Memory Update:

$$\mathbf{R}_{\ell} = \operatorname{Attn}_{P_{\ell},O_{\ell}}(\mathbf{Z}_{\ell}) + \gamma_{\ell}\mathbf{R}_{\ell-1}.$$

• Correspondence with CGD:

$$\mathbf{s}_n = \mathbf{r}_n + \gamma_n \mathbf{s}_{n-1}$$
.

Identifying  $\mathbf{R}_{\ell} \leftrightarrow \mathbf{s}_n$ ,  $\gamma_{\ell} = \gamma_n$ , and  $\mathbf{R}_{\ell-1} \leftrightarrow \mathbf{s}_{n-1}$ , the Transformer's memory update matches CGD.

#### **Step 3: Update Parameters**

• Transformer Parameter Update:

$$\mathbf{Z}_{\ell+1} = \mathbf{Z}_{\ell} + \alpha_{\ell} \frac{1}{n} \mathbf{R}_{\ell}.$$

• Correspondence with CGD:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \alpha_n \mathbf{s}_n.$$

The scaling factor  $\frac{1}{n}$  accounts for the gradient's scaling, consistent with the CGD update when considering the Hessian  $\mathbf{H} = \frac{1}{n} \mathbf{X} \mathbf{X}^{\top}$ .

Step 4: Conjugacy Coefficient  $\gamma_{\ell}$  and Step Size  $\alpha_{\ell}$ 

- CGD Computations: Scalar values computed based on residuals and the Hessian.
- Transformer Implementation:
  - $\gamma_{\ell}$  and  $\alpha_{\ell}$  are treated as parameters, ensuring structural correspondence.
  - The Transformer's architecture allows these as fixed or learnable parameters.

Therefore, under suitable parameter configurations, the memory-augmented Transformer can implement CGD, demonstrating the feasibility of using the Transformer's architecture to perform CGD-like updates.

#### A.3 Full Proof of Theorem 3.2

**Theorem.** A memory-augmented Transformer can implement k steps of Linear First-Order Methods (LFOMs) by maintaining memory registers across layers, where the update rules are:

$$\mathbf{R}_{\ell} = \operatorname{Attn}_{P_{\ell}, Q_{\ell}}(\mathbf{Z}_{\ell}), \tag{A.11}$$

$$\mathbf{Z}_{\ell+1} = \mathbf{Z}_{\ell} + \frac{1}{n} \sum_{i=0}^{\ell} \Gamma_j^{\ell} \odot \mathbf{R}_{j}, \tag{A.12}$$

where  $\Gamma_j^{\ell}$  governs the contribution of previous layers, and  $\odot$  is the Hadamard (element-wise) product for scaling.

Our goal is to show that the memory-augmented Transformer with updates given by equations (A.11) and (A.12) can implement k steps of an LFOM, whose general formulation is:

$$\mathbf{w}^{k+1} = \mathbf{w}^0 + \sum_{i=0}^k \Lambda_i^k \nabla f(\mathbf{w}^i),$$

where  $\Lambda_i^k$  are diagonal matrices that scale the gradients  $\nabla f(\mathbf{w}^i)$ .

We will proceed by establishing a correspondence between the variables and updates in the memory-augmented Transformer and those in the LFOM, and by showing that, under appropriate parameter settings, the Transformer updates replicate the LFOM updates.

The first order of business is to realize that, in the proof of Lemma 1 (A.1), the  $\mathbf{w}_{k+1}^{\mathrm{gd}}$  update rule (A.10) is a direct downstream consequence of the  $\mathbf{Z}_{\ell+1}$  update rule (2.5), under the parameterization given in equation (2.8).

Set  $\mathbf{R}_{\ell} = \operatorname{Attn}_{P_{\ell},Q_{\ell}}(\mathbf{Z}_{\ell})$  per (A.11). Then the consequence of the  $\mathbf{Z}_{\ell+1} = \mathbf{Z}_{\ell} + \frac{1}{n} \sum_{j=0}^{\ell} \Gamma_{j}^{\ell} \odot \mathbf{R}_{j}$  update rule is that each  $\operatorname{Attn}_{P_{j},Q_{j}}(\mathbf{Z}_{j})$  is coordinate-wise scaled by  $\Gamma_{j}^{\ell} \in \mathbb{R}^{(d+1)\times(n+1)}$ . But if  $\operatorname{Attn}_{P_{j},Q_{j}}(\mathbf{Z}_{j})$  is coordinate-wise scaled by  $\Gamma_{j}^{\ell}$ , then the  $\mathbf{Y}_{k+1}$  update rule in (A.1) now instead looks like  $\mathbf{Y}_{k+1} = \mathbf{Y}_{k} - \frac{1}{n} \sum_{j=0}^{k} \Gamma_{j}^{k} \Big|_{d+1} \odot (\mathbf{Y}_{k} M \mathbf{X}_{0}^{\top} A_{k} \mathbf{X}_{0})$ , where  $\Gamma_{j}^{k} \Big|_{d+1}$  denotes the (d+1)-th row of  $\Gamma_{j}^{k}$ . This is because, by definition,  $\mathbf{Y}_{i}$  is the (d+1)-th row of  $\mathbf{Z}_{i}$  (A.1).

From the basic  $\mathbf{Y}_k$  update rule in (A.1), the update formula for  $\mathbf{y}_{n+1}^{(k+1)}$  in (A.2) follows as a consequence. Except that now, this update formula will include a coordinate-wise scaling as well, which we will denote by  $\Lambda_j^k \in \mathbb{R}^d$ :

$$\mathbf{y}_{n+1}^{(k+1)} = \mathbf{y}_{n+1}^{(k)} - \frac{1}{n} \sum_{j=0}^{k} \langle (A_j \bar{\mathbf{X}}^\top \bar{\mathbf{Y}}_j) \odot \Lambda_j^k, \mathbf{x}_{n+1} \rangle,$$

which in turn leads to  $\theta_{k+1} = \theta_k - \frac{1}{n} \sum_{j=0}^k (A_j \bar{\mathbf{X}} \bar{\mathbf{X}}^\top (\theta_j + \mathbf{w}^*)) \odot \Lambda_j^k$  in place of (A.4) and  $\mathbf{w}_{k+1}^{\mathrm{gd}} = \mathbf{w}_k^{\mathrm{gd}} - \frac{1}{n} \sum_{j=0}^k A_j \nabla R_{\mathbf{w}^*}(\mathbf{w}_j^{\mathrm{gd}}) \odot \Lambda_j^k$  in place of (A.5). The negative signs can, of course, be incorporated within the  $\Lambda_i^k$ s.

If we simply rewrite  $\Lambda_j^k \in \mathbb{R}^d$  as a diagonal matrix in  $\mathbb{R}^{d \times d}$ , this setup then subsumes the case of diagonal preconditioners  $\Lambda_j^k \in \mathbb{R}^{d \times d}$  acting on the gradients  $\nabla R_{\mathbf{w}^*}(\mathbf{w}_j^{\mathrm{gd}})$ , which in the general form looks like:

$$\mathbf{w}_{k+1}^{\mathrm{gd}} = \mathbf{w}_0 + \sum_{i=0}^k \Lambda_i^k \nabla R_{\mathbf{w}^*}(\mathbf{w}_i^{\mathrm{gd}}).$$

where  $\Lambda_i^k$  are diagonal matrices.

*Note.* The memory-augmented Transformer performs exactly these updates in the special case when the preconditioners  $A_j$  are scalar multiples of the identity. If the preconditioners  $A_j$  are non-trivial, then this architecture performs "**LFOM-like**" algorithms that lie in a class richer than LFOMs (3.2).

#### A.4 Full Proof of Theorem 4.1

**Theorem.** Consider a multi-head Memformer with H heads, where each head h is parameterized by

$$\{P_{\ell}^{h}, Q_{\ell}^{h}, \Gamma_{\ell}^{h}\}_{\ell=1}^{L}$$

Suppose that in-context prompts  $\mathbf{Z}_0$  are drawn from a mixture of M Gaussian components,

$$\left\{\mathcal{N}(\mu_m, \Sigma_m)\right\}_{m=1}^M$$
.

Assume that, after training, for each mixture component  $\mathcal{N}(\mu_m, \Sigma_m)$ , there exists a head  $h_m$  whose parameters yield near-optimal first-order performance on data from that component. Then, for each component m, there exist scalar gating coefficients

$$\{\alpha_{m,h}\}_{h=1}^H$$

such that, whenever a new prompt  $\mathbf{Z}_0$  is drawn from  $\mathcal{N}(\mu_m, \Sigma_m)$ , where  $\mathbf{x}^{(i)} \sim \mathcal{N}(\mu_m, \Sigma_m)$  i.i.d., and targets  $\mathbf{w}^*$  from  $\mathcal{N}(\mu_m, \Sigma_m^{-1})$ , the Memformer can form the linear combination

$$\sum_{h=1}^{H} \alpha_{m,h} \operatorname{Head}_{h}(\mathbf{Z}_{0})$$

to achieve near-optimal performance for that component. In particular, if one sets

$$\alpha_{m,h_m} = 1$$
 and  $\alpha_{m,h} = 0$  for all  $h \neq h_m$ ,

then the multi-head Memformer yields the same near-optimal updates as head  $h_m$  alone.

*Proof.* For each head  $h \in \{1,...,H\}$ , let  $\operatorname{Head}_h(\mathbf{Z}_0)$  denote the final update (or prediction) produced *solely* by that head's parameters

$$\left\{P_{\ell}^{h},\ Q_{\ell}^{h},\ \Gamma_{\ell}^{h}\right\}_{\ell=1}^{L}$$

when presented with the in-context prompt  $\mathbf{Z}_0$ . By assumption, for each mixture component m, there is an associated head  $h_m$  that is near-optimal for data drawn from  $\mathcal{N}(\mu_m, \Sigma_m)$ .

We now define gating coefficients

$$\alpha_m = (\alpha_{m,1}, \alpha_{m,2}, \ldots, \alpha_{m,H}),$$

to form a linear combination of the H heads' outputs. Specifically, in a forward pass on a new prompt  $\mathbb{Z}_0$  sampled from  $\mathcal{N}(\mu_m, \Sigma_m)$ , the multi-head Memformer can produce

$$\sum_{h=1}^{H} \alpha_{m,h} \operatorname{Head}_{h}(\mathbf{Z}_{0}).$$

Choosing  $\alpha_{m,h_m} = 1$  and  $\alpha_{m,h} = 0$  for  $h \neq h_m$  ensures that the combined update coincides exactly with  $\operatorname{Head}_{h_m}(\mathbf{Z}_0)$ . Since  $h_m$  is by hypothesis near-optimal on that mixture component, the overall performance on  $\mathcal{N}(\mu_m, \Sigma_m)$  is likewise near-optimal. Repeating this argument for each m shows that every mixture component can be matched with the head that specializes in it.

#### A.5 Full Proof of Theorem 5.1

**Theorem.** Fix a class of LFOMs parameterized by  $\theta \in \Theta \subset \mathbb{R}^p$ , for instance via diagonal preconditioners  $\{\Lambda_i^k\}$ . Let each LFOM  $\theta$  map the in-context prompt  $\mathbf{Z}_0$  to a final prediction  $\hat{y}_{\theta}(\mathbf{Z}_0)$  for  $\mathbf{x}(n+1)$ . Define the in-context loss

$$\ell(\theta; \mathbf{Z}_0) = (\hat{y}_{\theta}(\mathbf{Z}_0) - \langle \mathbf{x}(n+1), \mathbf{w}^* \rangle)^2.$$

Suppose:

- (i) The domain of  $\mathbf{Z}_0$ , as drawn from distribution  $\mathcal{D}$ , is such that the squared loss  $\ell(\theta; \mathbf{Z}_0)$  is almost surely bounded by a constant L > 0 or is sub-Gaussian with scale  $\sigma^2$ .
- (ii) The parameter space  $\Theta$  is a compact subset of  $\mathbb{R}^p$ .
- (iii) The mapping  $\theta \mapsto \ell(\theta; \mathbf{Z}_0)$  is continuous (or Lipschitz) for all  $\mathbf{Z}_0$  in the support of  $\mathcal{D}$ .

Draw N i.i.d. prompts  $\{\mathbf{Z}_0^{(i)}\}_{i=1}^N \sim \mathcal{D}$ , and let

$$\widehat{\theta} = \arg\min_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^{N} \ell(\theta; \mathbf{Z}_{0}^{(i)}).$$

Then, with probability at least  $1 - \delta$  over the sample draw,

$$\mathbb{E}_{\mathbf{Z}_0 \sim \mathcal{D}} \big[ \ell(\widehat{\theta}; \mathbf{Z}_0) \big] \leq \min_{\theta \in \Theta} \mathbb{E}_{\mathbf{Z}_0 \sim \mathcal{D}} \big[ \ell(\theta; \mathbf{Z}_0) \big] + \epsilon(N, \delta),$$

where  $\epsilon(N,\delta) \to 0$  as  $N \to \infty$ . Thus the learned LFOM parameters  $\widehat{\theta}$  generalize well on new prompts from  $\mathcal{D}$ .

*Proof.* Define the population risk  $R(\theta) := \mathbb{E}_{\mathbf{Z}_0 \sim \mathcal{D}}[\ell(\theta; \mathbf{Z}_0)]$  and the empirical risk  $\widehat{R}_N(\theta) := \frac{1}{N} \sum_{i=1}^N \ell(\theta; \mathbf{Z}_0^{(i)})$ . The estimator  $\widehat{\theta}$  minimizes  $\widehat{R}_N(\theta)$  over the compact set  $\Theta \subset \mathbb{R}^p$ .

Because p is finite and the loss is bounded or sub-Gaussian, classical parametric uniform convergence guarantees apply. Concretely, by, for example, the standard Glivenko–Cantelli or

Vapnik–Chervonenkis theory for finite-dimensional parameter spaces (see, e.g., Shalev-Shwartz and Ben-David (2014) or Geer (2000)), there is a function  $\varepsilon(N,\delta)$  going to 0 as  $N\to\infty$  such that, with probability at least  $1-\delta$ , one has

$$\sup_{\theta \in \Theta} \left| |\widehat{R}_N(\theta) - R(\theta)| \right| \leq \epsilon(N, \delta).$$

Since  $\widehat{\theta}$  minimizes  $\widehat{R}_N(\theta)$ , a straightforward argument gives

$$R(\widehat{\theta}) = \widehat{R}_N(\widehat{\theta}) + (R(\widehat{\theta}) - \widehat{R}_N(\widehat{\theta})) \le \widehat{R}_N(\theta^*) + \epsilon(N, \delta),$$

where  $\theta^* \in \arg\min_{\theta \in \Theta} R(\theta)$ . Another  $\epsilon(N, \delta)$ -term bounds the difference between  $\widehat{R}_N(\theta^*)$  and  $R(\theta^*)$ , which yields

$$R(\widehat{\theta}) \leq R(\theta^*) + 2\epsilon(N, \delta).$$

Thus, possibly redefining  $\epsilon(N,\delta)$  by a small constant factor, we obtain the usual generalization guarantee

$$R(\widehat{\theta}) \leq \min_{\theta \in \Theta} R(\theta) + \epsilon(N, \delta).$$

Because  $R(\theta)$  is exactly the expected in-context loss in question, this completes the proof.

**Note.** The boundedness (or sub-Gaussianity) of  $\ell(\theta; \mathbf{Z}_0)$ , plus the compactness and continuity assumptions on  $\theta$ , are key to ensuring uniform convergence in finite-dimensional parameter spaces. Such conditions are typically satisfied in standard LFOM settings, e.g. when the data are bounded or sub-Gaussian and the square loss is used. If data or parameters are unbounded, one can impose appropriate norm constraints and regularity conditions to obtain similar results.

## B Comparison to Nesterov Accelerated Gradient Method (NAG) and Momentum Gradient Descent (MGD)

#### **B.1** Nesterov Accelerated Gradient Method (NAG)

NAG is a commonly used optimization technique that builds on classical gradient descent by incorporating a momentum term that anticipates the next update. Specifically, the weights are updated using the following update rules:

$$\mathbf{v}_{k+1} = \mathbf{w}_k + \beta_k (\mathbf{w}_k - \mathbf{w}_{k-1})$$

$$\mathbf{w}_{k+1} = \mathbf{v}_{k+1} - \eta_k \nabla f(\mathbf{v}_{k+1})$$

Here,  $\beta_k$  controls the influence of previous updates (momentum), and  $\eta_k$  is the learning rate. In our experiments, we selected  $\eta_k = 0.03$  and  $\beta_k = 0.9$  after testing various values of these parameters on the given distribution, as in Section 3.3. These values provided the best performance. The momentum term allows NAG to "look ahead" in the optimization trajectory, which often leads to faster convergence than vanilla gradient descent.

#### **B.2** Momentum Gradient Descent (MGD)

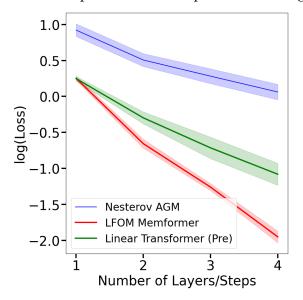
Momentum Gradient Descent operates similarly to NAG but without the anticipation of future steps. The algorithm updates the weights based on a momentum term that accelerates convergence in directions with consistent gradients. The update rule for MGD is given by:

$$\mathbf{v}_{k+1} = \beta_k \mathbf{v}_k - \eta_k \nabla f(\mathbf{w}_k)$$
$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{v}_{k+1}$$

In our experiments, the learning rate  $\eta_k = 0.005$  and momentum parameter  $\beta_k = 0.9$  provided the best results on the given distribution, as in Section 3.3. Momentum helps to mitigate oscillations in directions with high curvature, stabilizing the optimization trajectory and leading to faster convergence compared to gradient descent.

#### **B.3** Memformers vs. NAG and MGD

In our experiments, we observed that Memformers (3.4) outperform both NAG and MGD. Figures 12 and 13 compare the performance of Memformer with NAG and MGD, respectively, on the same non-isotropic data. As shown, the Memformer achieves faster convergence and much better loss performance compared to both algorithms.



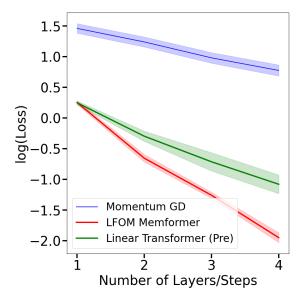


Figure 12: Nesterov AGM vs. LFOM Memformer

Figure 13: Momentum GD vs. LFOM Memformer

Figure 14: Comparison of LFOM Memformer with (a) Nesterov AGM and (b) Momentum GD.

## C LFOM Memformer Experiments With More Than 4 Layers

In our experiments, we observed that Memformers with more than 4 layers continue to demonstrate impressive performance in learning optimization strategies. We conducted experiments with Memformers having up to 7 layers and dimension d = 10. Training beyond this point becomes

impractical due to extensive iteration requirements and significant convergence times, which can span several hours. This limitation is a consequence of computational constraints (e.g., available GPUs) rather than any inherent deficiency of the Memformer architecture itself. Here, d refers to the rank of the square matrix  $XX^T$  in the empirical loss quadratic as described in Equation 2.11.

- 1. **Experiment (Figure 15)** (Dimension d = 5, Layers = 5): As expected, Conjugate Gradient Descent (CGD) converges within d steps due to the dimensionality constraint. Remarkably, even though the Memformer only learns general parameters  $\mathbf{A}_{\ell}$  (Equation (2.8)) and  $\Gamma_{\ell}$  (Equation (3.4)), it manages to keep up with CGD for up to 4 steps, showcasing its efficiency.
- 2. Experiment (Figure 16) (Dimension d = 10, Layers = 7): In this case, CGD does not converge until beyond 7 steps, which aligns with theoretical expectations. Nevertheless, the Memformer remains highly competitive, matching CGD's performance for 6 steps and even performing comparably at 7 steps. This demonstrates the Memformer's robust generalization capabilities, even under more complex conditions.

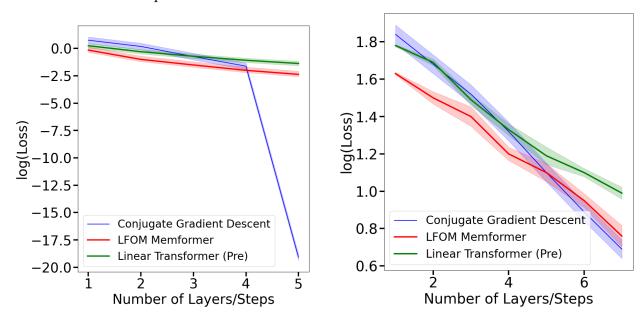


Figure 15: Memformer performance for d = 5 with 5 layers.

Figure 16: Memformer performance for d = 10 with 7 layers.

Figure 17: Comparison of Memformer performance for different configurations of depth *d* and layers.

## D Experiment on Convergence Verification for Memformer Parameter $A_{\ell}$ to $\Sigma$

Our strategy to train the Memformer (3.4) was to first train the  $A_{\ell}$ 's (2.8) in each layer  $\ell$  on the training batch and then to "fine-tune" the  $\Gamma_{\ell}$ 's on the training batch. Therefore, we present here an empirical verification of our results per **Theorem 3** in Ahn et al. (2024).

**Theorem 3. (Ahn et al. (2024))** Assume that  $x^{(i)} \stackrel{iid}{\sim} \mathcal{N}(0,\Sigma)$  and  $w_x \sim \mathcal{N}(0,\Sigma^{-1})$ , for  $i=1,\ldots,n$ , and for some  $\Sigma \succ 0$ . Consider the optimization of in-context loss (2.7) for a k-layer transformer with the

parameter configuration in Eq. (2.8) given by:

$$\min_{\{A_{\ell}\}_{\ell=0}^{L-1}} f(A).$$

Let  $S \subset \mathbb{R}^{L \times d \times d}$  be defined as follows:  $A \in S$  if and only if for all i = 0, ..., L - 1, there exist scalars  $a_i \in \mathbb{R}$  such that  $A_i = a_i \Sigma^{-1}$ . Then

$$\inf_{(A,B)\in S} \sum_{i=0}^{L-1} \|\nabla_{A_i} f(A,B)\|_F^2 = 0,$$

where  $\nabla_{A_i} f$  denotes the derivative with respect to the Frobenius norm  $||A_i||_F$ .

We evaluated the in-context learning (ICL) loss for linear regression with d=5 and n=20, where  $x^{(i)} \sim \mathcal{N}(0,\Sigma)$  and  $w_x \sim \mathcal{N}(0,\Sigma^{-1})$ . The covariance  $\Sigma$  was generated as  $\Sigma = U^T D U$ , with U being a random orthogonal matrix and  $D = \operatorname{diag}(1,1,1/4,1/16,1)$ . A three-layer linear transformer was trained using ADAM, with  $A_0$ ,  $A_1$ ,  $A_2$  initialized as i.i.d. Gaussian matrices. Each gradient step used minibatches of size 20,000, resampled every 100 steps, and gradients were clipped to 0.01. Results were averaged over 5 runs with independent U and  $\Sigma$  samples.

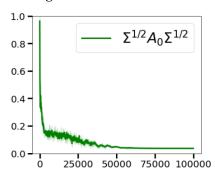
To measure convergence, we computed the normalized Frobenius norm distance:

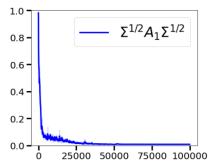
$$Dist(M, I) := \min_{\alpha} \frac{\|M - \alpha I\|_F}{\|M\|_F},$$

where

$$\alpha := \frac{1}{d} \sum_{i=1}^{d} M[i, i],$$

which quantifies the deviation of  $M/\|M\|_F$  from a scaled identity. The distance  $\text{Dist}(\Sigma^{1/2}A_i\Sigma^{1/2}, I)$ , averaged over 5 runs, is shown in Figures 18, 19, and 20 as a function of training iterations.





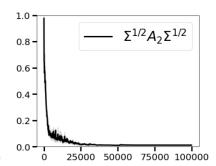


Figure 18:  $A_0$  convergence.

Figure 19:  $A_1$  convergence.

Figure 20:  $A_2$  convergence.

Figure 21: Convergence comparison of  $A_0$ ,  $A_1$ , and  $A_2$ .