TPTT: Transforming Pretrained Transformers into Titans

Fabien Furfaro[*]

2025

Abstract

Background: Transformer-based large language models (LLMs) have shown capabilities across various natural language processing tasks. However, their quadratic computational and memory complexity, especially in self-attention, challenges efficient inference on long contexts and deployment in limited-resource environments.

Methods: We introduce TPTT (Transforming Pretrained Transformers into Titans), a framework that augments pretrained Transformers with linearized attention and internal memory gating via Memory as Gate (MaG), without full retraining. TPTT supports parameter-efficient fine-tuning (LoRA) and integrates with standard libraries like Hugging Face Transformers.

Results: Experiments on pretrained models of about 1 billion parameters, primarily on the MMLU benchmark, suggest improvements in both efficiency and accuracy over baselines. For instance, Titans-Llama-3.2-1B shows up to 20% relative gain in Exact Match scores in one-shot evaluation. Training uses modest computational resources, supporting practical applicability.

Conclusions: These preliminary results indicate that TPTT may aid adaptation of pretrained LLMs to long-context tasks with limited overhead. Further evaluation on larger models and diverse benchmarks is needed to assess generality. Source code and package are available at `https://github.com/fabienfrfr/tptt` and `https://pypi.org/project/tptt/`.

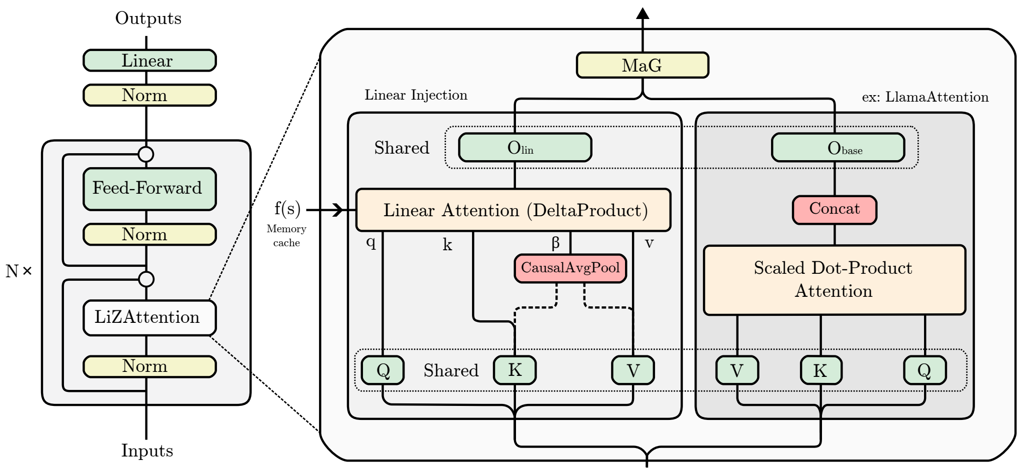

Figure 1: Overview of the TPTT architecture. On the left, the diagram illustrates a decoder-only architecture where linear attention is injected in parallel of vanilla attention (LiZAttention). On the right, the detailed architecture of the linearized attention mechanism is depicted, highlighting the shared weights for query (Q), key (K), value (V), and output (O) projections. It also shows the management of the state memory (S) and the combination of outputs through the Memory as Gate (MaG) weighting mechanism. The state can be unlinear between chunk. The diagram emphasizes the integration of linearized attention mechanisms and advanced memory management techniques, such as DeltaProduct and AdaptativeAvgPool1D, contributing to processing and output generation.

1 Introduction

Transformer-based large language models (LLMs) have revolutionized natural language processing (NLP), achieving state-of-the-art results across a wide range of tasks, from text generation to code synthesis [11, 19].

[*] `fabien.furfaro@gmail.com`

However, the quadratic complexity of the self-attention mechanism in both computation and memory remains a major limitation, particularly for applications requiring long-context inference, such as document understanding, multi-turn dialogue, and scientific reasoning.

Recent research has sought to address these challenges through a variety of approaches. Efficient attention mechanisms [9, 21] aim to reduce the computational burden by approximating or restructuring the attention operation, while recurrent and memory-augmented architectures [2, 13] introduce internal memory to better capture long-range dependencies. At the same time, parameter-efficient fine-tuning techniques such as LoRA [8] have made it feasible to adapt large pretrained models to new domains or tasks without the need for full retraining.

Despite these advances, most existing solutions require substantial architectural changes or retraining from scratch, limiting their applicability to already deployed or proprietary models. There remains a need for methods that can efficiently endow existing LLMs with long-context and memory-augmented capabilities, while preserving their pretrained knowledge and minimizing adaptation cost.

In this paper, we introduce TPTT (Transforming Pretrained Transformers into Titans), a practical and scalable framework for upgrading pretrained Transformers with efficient linearized attention and advanced memory mechanisms. Our approach integrates techniques such as Memory as Gate (MaG) and mixed linearized attention (LiZA), inspired by recent advances in memory-augmented neural architectures [2]. TPTT is fully compatible with the Hugging Face Transformers library and leverages parameter-efficient fine-tuning (LoRA), enabling seamless adaptation of any causal LLM to long-context tasks.

Our main contributions are:

- We propose TPTT, a framework for augmenting pretrained Transformers with linearized attention and memory gating, requiring minimal changes to the original architecture.
- We introduce the MaG mechanism, which adaptively combines linear and softmax attention outputs, and demonstrate its effectiveness for long-context modeling.
- We provide an open-source implementation compatible with popular frameworks, and show that TPTT enables substantial improvements in both efficiency and accuracy on challenging benchmarks such as MMLU.

The remainder of this paper is organized as follows: Section **??** reviews related work. Section **??** details the TPTT framework and its components. Section **??** presents experimental results and analysis. Section **??** discusses limitations and future directions, and Section **??** concludes.

2 Related Work

The computational and memory challenges of Transformer models have motivated a rich body of research aimed at improving efficiency and scalability.

Efficient Attention Mechanisms Methods such as FlashAttention [3] and Mamba [6] optimize the implementation of the standard attention mechanism to reduce memory footprint and improve throughput, often via kernel fusion, tiling, or custom CUDA kernels. While these approaches accelerate inference and training, they do not fundamentally alter the quadratic complexity with respect to sequence length.

Linearized and Approximated Attention To address the quadratic bottleneck, various linear attention variants have been proposed. Linformer [20] approximates attention by low-rank projections, Performer [**?**] uses random feature maps to linearize the softmax kernel, and LoLCat [22] and Liger [10] further explore kernel-based and structural approximations. However, these methods often require retraining from scratch or significant architectural modifications, limiting their applicability to pretrained models. Notably, Liger exploits similar properties to those leveraged in this work, but does not rely on explicit linearization injection (and use only less expressive Gated Linear Attention [**?**] then DeltaProduct (this work) but more fast).

Recurrent and Memory-Augmented Architectures Alternative designs such as RWKV [**?**] and Titans [2] introduce recurrent or memory-augmented components to capture long-range dependencies efficiently. DeltaProduct [16] extends this idea by leveraging products of Householder transformations to enhance expressivity while maintaining stability. These architectures demonstrate promising results but typically require training models from the ground up.

Parameter-Efficient Fine-Tuning Techniques like LoRA [8] and adapters enable adapting large pretrained models to new tasks with a small number of trainable parameters, reducing computational cost and preserving pretrained knowledge. However, integrating efficient attention or memory mechanisms into pretrained models via parameter-efficient fine-tuning remains underexplored.

Our Positioning In contrast to prior work, TPTT offers a practical framework to augment pretrained Transformers with efficient linearized attention and advanced memory gating, requiring minimal architectural changes and enabling seamless integration with popular libraries such as Hugging Face Transformers. By combining Memory as Gate (MaG) and mixed linearized attention (LiZA), TPTT facilitates long-context adaptation through parameter-efficient fine-tuning, unlocking new capabilities for existing LLMs without full retraining.

3 Methodology

3.1 Linearized Attention Mechanisms

Standard self-attention in transformers computes pairwise interactions between all tokens, resulting in quadratic complexity with respect to sequence length (i.e., $O(T^2D)$ where T is the sequence length and D the feature dimension) [19]. To address this computational bottleneck, linearized attention mechanisms approximate the softmax attention by projecting queries and keys through a carefully chosen feature map ϕ [9,13,20,21]. Typical choices for ϕ include kernel feature maps or nonlinearities such as ELU, which enable rewriting the attention as a series of linear operations. This approximation reduces both computational and memory costs, allowing efficient processing of long sequences while maintaining substantial modeling power [3,6,10,22].

However, linearized attention approaches can suffer from reduced expressivity compared to full softmax attention, especially in capturing complex, non-linear dependencies across tokens. This motivates the development of mechanisms that balance efficiency and expressivity.

Formally, given an input sequence $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\} \in \mathbb{R}^{T\times D}$, the standard self-attention computes:

$$\mathbf{Q}, \mathbf{K}, \mathbf{V} = \mathbf{X}\mathbf{W}_Q, \mathbf{X}\mathbf{W}_K, \mathbf{X}\mathbf{W}_V \tag{1}$$

$$\mathbf{O}_{\text{base}} = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{D}}\right)\mathbf{V} \tag{2}$$

In linear attention, the softmax kernel is approximated by a feature mapping ϕ, yielding the output at position t:

$$\mathbf{O}_{\text{lin},(t)} = \frac{\phi(\mathbf{q}_t)^\top \left(\sum_{i=1}^{t} \phi(\mathbf{k}_i)\beta_i\, \mathbf{v}_i^\top\right)}{\phi(\mathbf{q}_t)^\top \left(\sum_{i=1}^{t} \phi(\mathbf{k}_i)\beta_i\right)} = \frac{\phi(\mathbf{q}_t)^\top S_t}{\phi(\mathbf{q}_t)^\top z_t} \tag{3}$$

where $\mathbf{q}_t$, $\mathbf{k}_i$, and $\mathbf{v}_i$ are the query, key, and value vectors at positions t and i, respectively, and β_i is a gating scalar or vector modulating the keys and values. Here, β_i is applied multiplicatively after the feature mapping ϕ, which itself can be an ELU-based or kernel approximation function [20].

In this work, β_i is constructed via causal average pooling over the keys and values, which helps stabilize the scaling of the attention scores over long sequences. And we apply apply SiLU normalized in Q,K,V, RMSnorm for output and sigmoid for beta.

$$\mathbf{X} \leftarrow \phi(\mathbf{S}_{[t,\,t+C-1]}) \tag{4}$$

$$\beta \leftarrow \phi(\mathbf{S}_{[t,\,t+C-1]}) \tag{5}$$

$$\mathbf{Y} \leftarrow \phi(\mathbf{S}_{[t,\,t+C-1]}) \tag{6}$$

Where X is Q,K,V.

3.2 Memory as Gate (MaG)

To further improve modeling of long-range dependencies, we introduce *Memory as Gate* (MaG), an internal memory augmentation mechanism inspired by Titans [2]. MaG enables the model to dynamically combine the strengths of linearized and softmax attention, adapting to the context and sequence characteristics.

Given the outputs of linearized attention $\mathbf{O}_{\text{lin}}$ and softmax attention $\mathbf{O}_{\text{base}}$, MaG computes the final attention output as two strategy:

1. **Gated Mixing:** A learnable parameter α is broadcast to match the attention tensor shapes. The outputs are mixed as:

$$\mathbf{O}_{\text{MaG}} = (1-\alpha)\cdot\mathbf{O}_{\text{base}} + \alpha\cdot\mathbf{O}_{\text{lin}} \tag{7}$$

2. **Cross-Gate Mixing:** Optionally, MaG can introduce a nonlinear interaction term between the two attention outputs:

$$\mathbf{O}_{\text{MaG}} = (1-\alpha)\cdot\mathbf{O}_{\text{base}} + \alpha\cdot\mathbf{O}_{\text{lin}} + [(1-\alpha)\cdot\mathbf{O}_{\text{base}}]\odot[\alpha\cdot\mathbf{O}_{\text{lin}}] \tag{8}$$

where $\alpha \in [0,1]$ is a learnable parameter adapted during training and $\odot$ denotes elementwise multiplication. This "cross-gate" term enables richer, nonlinear mixing between the two attention mechanisms, potentially capturing more complex dependencies (see also [15]). In practice, α can be a scalar per layer or per attention head, allowing flexible weighting between efficiency (linear attention) and expressivity (softmax attention). This gating mechanism can be interpreted as a form of memory-augmented control, where the model learns to rely more or less on the efficient approximation depending on the input context.

3.3 Parallel Delta Product Modeling

We leverage the **DeltaProduct** operator as the feature mapping ϕ within the linear attention framework, generalizing the classical Delta rule to increase expressivity while preserving efficient parallel computation. DeltaProduct addresses the expressivity bottleneck found in linear recurrent architectures that rely on diagonal or diagonal-plus-rank-1 state transitions by employing a product of multiple generalized Householder transformations per token [16]. This structure also ensures numerical stability due to the orthogonality properties of Householder matrices.

3.3.1 Delta Rule Formulation

Sequential Delta Rule. The classical Delta rule for associative memory updates the state matrix $\mathbf{S}_t \in \mathbb{R}^{d\times d}$ at time t as:

$$\begin{aligned}\mathbf{S}_t &= \mathbf{S}_{t-1}\left(\mathbf{I} - \beta_t\mathbf{k}_t\mathbf{k}_t^\top\right) + \beta_t\mathbf{v}_t\mathbf{k}_t^\top && (9)\\ &= \mathbf{S}_{t-1} + \beta_t\left(\mathbf{v}_t - \mathbf{S}_{t-1}\mathbf{k}_t\right)\mathbf{k}_t^\top && (10)\end{aligned}$$

where $\mathbf{k}_t, \mathbf{v}_t \in \mathbb{R}^d$ are the key and value vectors, and β_t is a gating scalar. This update is inherently sequential and thus not well-suited for parallel hardware acceleration.

Expanded recursive form. Expanding the recursion, the state $\mathbf{S}_t$ can be expressed as the cumulative effect of all past updates modulated by subsequent corrections, i.e.,

$$\mathbf{S}_t = \sum_{i=1}^{t}\beta_i\mathbf{v}_i\mathbf{k}_i^\top\prod_{j=i+1}^{t}\left(\mathbf{I} - \beta_j\mathbf{k}_j\mathbf{k}_j^\top\right), \tag{11}$$

which highlights how each rank-1 update is successively transformed by the multiplicative decay terms from later steps.

Chunkwise Parallelization. To enable parallel computation, the input sequence is partitioned into chunks of size C. The update within each chunk is computed in parallel as [21]:

$$\begin{aligned}\mathbf{S}_{[t,\,t+C-1]} &= \mathbf{S}_{t-1} + \mathbf{K}^\top\mathbf{Y} && (12)\\ \mathbf{Y} &= \mathbf{T}^{-1}\mathbf{R} && (13)\\ \mathbf{R} &= \mathbf{V}\odot\boldsymbol{\beta} - (\mathbf{K}\odot\boldsymbol{\beta})\mathbf{S}_{t-1} && (14)\\ \mathbf{T} &= \mathbf{I} - \operatorname{tril}\left((\mathbf{K}\odot\boldsymbol{\beta})\mathbf{K}^\top, -1\right) && (15)\end{aligned}$$

where $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{C\times d}$ are stacked keys and values for the chunk, $\boldsymbol{\beta} \in \mathbb{R}^{C\times 1}$ is the gating vector, $\odot$ denotes elementwise multiplication, and with $\operatorname{tril}(\cdot, -1)$ extracting the strictly lower triangular part. This formulation enables efficient batched computation by restricting dependencies within chunks.

3.3.2 Delta Product

DeltaProduct: Higher-Order Parallel State Updates. DeltaProduct generalizes the above by applying n_h rank-1 Householder-like updates per token. The state update at position i is recursively defined as:

$$\mathbf{H}_{i,j} = \mathbf{H}_{i,j-1} + \beta_{i,j}\left(\mathbf{v}_{i,j} - \mathbf{H}_{i,j-1}\mathbf{k}_{i,j}\right)\mathbf{k}_{i,j}^\top, \quad j = 1,\ldots,n_h \tag{16}$$

with $\mathbf{H}_{i,0} = \mathbf{H}_{i-1}$ and $\mathbf{H}_{i,n_h} = \mathbf{H}_i$. Each $\mathbf{k}_{i,j}, \mathbf{v}_{i,j}, \beta_{i,j}$ is generated from the input at position i for the j-th Householder step, typically via independent learned linear projections.

Expanded recursive form. By unrolling the recursive updates over the internal steps $j = 1, \dots, n_h$ for token i, the state $\mathbf{H}_i$ can be equivalently expressed as:

$$\mathbf{H}_i = \prod_{j=1}^{n_h} \left(\mathbf{I} - \beta_{i,j}\mathbf{k}_{i,j}\mathbf{k}_{i,j}^\top\right) \mathbf{H}_{i-1} + \sum_{j=1}^{n_h} \left(\beta_{i,j}\mathbf{k}_{i,j}\mathbf{v}_{i,j}^\top \prod_{\ell=j+1}^{n_h} \left(\mathbf{I} - \beta_{i,\ell}\mathbf{k}_{i,\ell}\mathbf{k}_{i,\ell}^\top\right) \right) \tag{17}$$

where: - The product $\prod_{j=1}^{n_h}(\cdot)$ denotes the ordered matrix product (applied from right to left). - The first term corresponds to the state transition matrix obtained by chaining n_h generalized Householder reflections applied to the previous token's state $\mathbf{H}_{i-1}$. - The second term sums the rank-1 corrections given by the values $\mathbf{v}_{i,j}$, each transformed by subsequent Householder matrices.

This expansion reveals that DeltaProduct's transition matrix is a product of n_h such transformations, interpolating between simple DeltaNet (where $n_h = 1$) and richer, more expressive state transitions. It explicitly encodes how multiple gradient steps per token compose to update the memory state.

Chunkwise DeltaProduct Update. For a chunk of C real tokens (expanded to $N = n_h C$ virtual tokens), the parallel state update is:

$$\mathbf{T}_n = \mathbf{I}_N - \operatorname{tril}\left((\mathbf{K}_n \odot \boldsymbol{\beta}_n)\mathbf{K}_n^\top, -1\right) \tag{18}$$

$$\mathbf{U}_n = \mathbf{V}_n \odot \boldsymbol{\beta}_n - (\mathbf{K}_n \odot \boldsymbol{\beta}_n)\mathbf{S}_{\text{in}} \tag{19}$$

$$\mathbf{y}_n = \mathbf{T}_n^{-1}\mathbf{U}_n \tag{20}$$

$$\mathbf{S}_{[t,\, t+C-1]} = \mathbf{S}_{\text{in}} + \mathbf{K}_n^\top \mathbf{y}_n \tag{21}$$

where $\mathbf{K}_n, \mathbf{V}_n \in \mathbb{R}^{N\times d}$ and $\boldsymbol{\beta}_n \in \mathbb{R}^{N\times 1}$ are the keys, values, and gates for all virtual tokens in the chunk, and $\mathbf{S}_{\text{in}}$ is the initial state for the chunk (typically the last state of the previous chunk).

Virtual Token Expansion: Motivation and Formulation. To enable multiple Householder updates per token in parallel, each real token $\mathbf{x}_t \in \mathbb{R}^D$ is expanded into n_h virtual tokens using strategies designed to enrich expressivity while maintaining numerical stability. We have two types of transformations:

$$\text{Derivative trick:} \quad \mathbf{x}_{t,\text{deriv}}^{(m)} = \frac{1}{Z_m}\sum_{k=0}^{m}(-1)^k \binom{m}{k} \mathbf{x}_{t-k}, \quad m = 0, \dots, n_h - 1, \tag{22}$$

$$\text{Rotary trick:} \quad \mathbf{x}_{t,\text{rot}}^{(m)} = \mathbf{R}(\theta_m)\mathbf{x}_t, \quad \text{with} \quad \theta_m = \frac{2\pi m}{n_h}, \tag{23}$$

where $\mathbf{R}(\theta_m)$ applies a phase rotation of angle θ_m to consecutive pairs of features of $\mathbf{x}_t$, and Z_m normalizes the scale. This expansion is applied independently to each input sequence $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$, and $\boldsymbol{\beta}$, resulting in $N = n_h C$ virtual tokens per chunk.

Nonlinearity. Optionally, a nonlinearity ϕ (e.g., GELU, tanh) can be applied to the state update at each chunk boundary or after the parallel update:

$$\mathbf{S}_{[t,\, t+C-1]} \leftarrow \phi(\mathbf{S}_{[t,\, t+C-1]}) \tag{24}$$

This is controlled by a flag in the implementation and can improve model capacity and stability.

Expressivity and Efficiency. By increasing the order n, DeltaProduct interpolates between the original DeltaNet ($n = 1$) and a dense state transition. Notably, DeltaProduct of order $n = 2$ achieves a level of expressivity comparable to the Titans model and can solve algorithmic problems beyond the class TC^0 (such as parity) [14, 16]. This is in contrast to diagonal or single-rank updates, which are fundamentally limited in this regard. DeltaProduct thus provides a tunable trade-off between efficiency and expressivity, while retaining the hardware efficiency of the chunkwise parallel algorithm.

3.4 Integration with Pretrained Models

Our approach injects linearized attention and memory augmentation modules into pretrained Transformers models. The process involves:

1. **Identification of Target Modules:** Key attention layers to be modified are identified using tools such as *get_ tptt_ model.*

2. **Modification of Attention Layers:** These layers are replaced or extended with the proposed *LiZAttention* module, which implements both linear and softmax attention with linear projection weigth sharing and MaG.

3. **Training and Fine-Tuning:** The modified model is fine-tuned using parameter-efficient methods such as LoRA [8], ensuring optimal adaptation to the new mechanisms without full retraining.

This procedure enables the transformation of any causal pretrained LLM into a memory-augmented, efficient architecture with minimal overhead, that without any new layers.

3.5 LiZAttention Module

The `LiZAttention` module is a core component of the TPTT architecture, designed to synergistically combine linearized attention and standard (softmax) attention mechanisms. This hybrid approach leverages the computational efficiency of linear attention while retaining the expressivity of vanilla attention. To support long-context inference, `LiZAttention` maintains a cache of intermediate states and implements a recurrent information mechanism for efficient internal memory management [9].

Algorithm 1 LiZAttention Forward Pass

```
Require: hidden_states $\in \mathbb{R}^{B\times L\times D}$          ▷ Batch size $B$, sequence length $L$, embedding dim $D$
Require: mask                                       ▷ Attention mask for padding/causality
1: Projection:
     Compute queries $q$, keys $k$, values $v$ via learned projections:
     $q \in \mathbb{R}^{B\times H\times L\times d_h}$, $k, v \in \mathbb{R}^{B\times H_k\times L\times d_h}$
2: Apply Attention Mask:
     Apply mask to $k$ and $v$ to handle padding and restrict attention.
3: Linear Attention:
     Compute linear attention output $o_{\text{lin}}$ using a feature map $\phi$:
     $o_{\text{lin}}[t] = \frac{\sum_{i=1}^{t} \phi(q_t)^\top \phi(k_i) v_i}{\sum_{i=1}^{t} \phi(q_t)^\top \phi(k_i)}$
     Store intermediate states in a memory cache for recurrent information.
4: Vanilla (Softmax) Attention:
     Compute standard self-attention output $o_{\text{base}}$ (optionally truncated for long sequences):
     $o_{\text{base}} = \text{Softmax}\left(\frac{qk^\top}{\sqrt{d_h}}\right) v$
5: Combine Outputs (Memory as Gate):
     Compute final output using a learnable gating parameter $\alpha$:
     $o = \alpha \cdot o_{\text{lin}} + (1-\alpha) \cdot o_{\text{base}}$
6: return $o \in \mathbb{R}^{B\times L\times D}$
```

3.5.1 Efficient Internal Memory Management

The cache of intermediate states maintained by `LiZAttention` enables a recurrent information, efficiently supporting long-context inference without excessive computational overhead [9]. This approach allows the model to scale to longer sequences, leveraging both local and global context.

4 Training Procedure

4.1 Parameter-Efficient Fine-Tuning with LoRA

To adapt the TPTT architecture to downstream tasks, we employ Low-Rank Adaptation (LoRA) [4, 8], a parameter-efficient fine-tuning technique that injects trainable low-rank matrices into selected projection layers while freezing the original model weights. This approach reduces the number of trainable parameters and memory requirements, while maintaining performance comparable to full fine-tuning [4, 8]. LoRA is configured with a rank of 8, $\alpha = 16$, and a dropout rate of 0.05. Fine-tuning targets the main projection modules, specifically `[q_proj, k_proj, v_proj, o_proj]` for Llama/Mistral and `[qkv_proj, out_proj]` for OpenELM [4].

4.2 Dynamic Memory as Gate Scheduling (LiZA Callback)

A important component of the training process is the LiZA callback, which dynamically adjusts the Memory as Gate (MaG) weighting parameter during training. The MaG weight is initialized at 0.01 and linearly increased to 0.5 over the first 100 steps, facilitating a smooth transition from reliance on vanilla (softmax) attention to linearized attention. This schedule allows the model to effectively balance the two attention mechanisms, optimizing performance throughout training. The callback is integrated directly into the training loop, ensuring adaptive control of the MaG parameter and enhancing the model's adaptability and efficiency.

5 Experiments and Results

5.1 Experimental Setup

We evaluated the TPTT library on several pretrained language models with approximately 1 billion parameters, using the MMLU benchmark [7] as the primary evaluation suite. Training was conducted on 500 samples from the `yahma/alpaca-cleaned` dataset [17] for 5 epochs, with a maximum sequence length of 384 tokens, a batch size of 3, and a learning rate of 5×10^{-4}. Mixed precision training and gradient clipping at 1.0 were employed to optimize computational efficiency and stability. All experiments were performed on NVIDIA Tesla T4 GPUs (Kaggle platform). The trained models and detailed metrics are publicly available on the Hugging Face Model Hub[1], with full training logs accessible via Hugging Face TensorBoard.

5.2 Training Results

Table 1 summarizes the training performance metrics for various TPTT models. The results indicate consistent and efficient learning across architectures, by low final loss values and stable gradient norms. The use of Low-Rank Adaptation (LoRA) and the Memory as Gate (MaG) mechanism shows effective in optimizing training dynamics and convergence.

Model	Loss	Training Time (s)	Samples/s	Steps/s	Total FLOPs	Gradient Norm	Refs Based On
Titans-Llama-3.2-1B	1.375	1654.1	1.51	0.254	5.62e18	2.68	[18]
Titans-OpenELM-1_1B	1.3188	1651.1	1.51	0.254	5.85e18	0.704	[12]
Titans-Qwen2.5-1.5B	1.2568	1900.6	1.31	0.221	7.56e18	1.99	[1]
Titans-OLMo-1B-hf	1.3068	1585.2	1.58	0.265	6.20e18	3.12	[5]

Table 1: Training performance metrics for TPTT models.

5.3 Callback comparizon Results

5.4 Evaluation Metrics and Benchmark Results

For evaluation, we focus on metrics standard in LLM and QA benchmarking: Exact Match (EM), Partial Exact Match (PEM), and Partial Quasi Exact Match (PQEM). These metrics respectively measure strict correctness, partial overlap, and quasi-exactness between model outputs and ground truth answers, providing a nuanced view of model performance [7].

Table 2 presents the MMLU benchmark results in the one-shot setting. TPTT models, and especially Titans-Llama-3.2-1B, consistently outperform their base counterparts in EM, with better PEM and PQEM scores. This shows the benefit of integrating linearized attention and internal memory mechanisms for complex language understanding tasks.

Model	EM $\pm$ Std	PEM $\pm$ Std	PQEM $\pm$ Std
Titans-Llama-3.2-1B	0.2456 ± 0.1276	0.2649 ± 0.1340	0.4772 ± 0.1569
Llama-3.2-1B	0.0070 ± 0.0058	0.3105 ± 0.1411	0.4719 ± 0.1530
Titans-Qwen2.5-1.5B	0.0000 ± 0.0000	0.5000 ± 0.1504	0.5825 ± 0.1442
Qwen2.5-1.5B	0.0000 ± 0.0000	0.5982 ± 0.1422	0.6895 ± 0.1288
Titans-OLMo-1B-hf	0.0000 ± 0.0000	0.2614 ± 0.1312	0.4649 ± 0.1540
OLMo-1B-hf	0.0000 ± 0.0000	0.2333 ± 0.1302	0.4246 ± 0.1533

Table 2: MMLU benchmark results (one-shot) with statistical analysis. Each pair groups a Titans model and its base counterpart.

[1] `https://huggingface.co/ffurfaro/`

5.5 Discussion and Comparison

Compared to recent state-of-the-art methods such as Mamba [6], LoLCat [22], and Liger [10], TPTT stands out by enabling the transformation of existing pretrained models without full retraining, while maintaining good benchmark performance. The observed improvements in EM and better PEM/PQEM scores highlight the effectiveness of TPTT's linearized attention and memory augmentation for efficient and robust LLM adaptation. These results confirm that TPTT is a practical and scalable solution for enhancing pretrained LLMs, especially in resource-constrained settings.

6 Discussion and Conclusion

In this paper, we have introduced TPTT, a novel framework for enhancing pretrained Transformers models by integrating efficient linearized attention mechanisms and internal memory augmentation. Our approach leverages parameter-efficient fine-tuning (LoRA) [8] to enable the rapid adaptation of large language models (LLMs) to long-context tasks, without the need for full retraining. Experimental results on the MMLU benchmark [7] shows significant improvements in both efficiency and accuracy, with robust statistical analyses and favorable comparisons to state-of-the-art methods [6, 10, 22].

Practical Implications. TPTT offers a scalable and practical solution for deploying high-performance LLMs in resource-constrained environments. The integration of linearized attention and memory augmentation reduces computational and memory requirements, making advanced language models more accessible for real-world applications. The use of LoRA allows for efficient and flexible fine-tuning, enabling rapid adaptation to new tasks and domains.

Limitations. Our current evaluation is limited to models of moderate size (around 1 billion parameters). Scaling TPTT to larger architectures and more diverse tasks may introduce new challenges, including increased tuning complexity and the need for further optimization of memory mechanisms. While our results are promising, broader validation on additional benchmarks and real-world scenarios is needed to fully assess the generalizability and robustness of the approach.

Future Directions. Future work will focus on optimizing the integration process, exploring more sophisticated internal memory mechanisms [2], and extending the evaluation to larger models and a wider range of benchmarks. Additional research will investigate hybrid approaches and the interplay between linearized attention, memory augmentation, and other efficiency-oriented techniques.

In summary, TPTT provides a practical, scalable, and effective library for upgrading pretrained Transformers, with strong empirical results and promising implications for the future of efficient language modeling.

References

[1] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

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

[3] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[4] Hugging Face. Lora - hugging face peft documentation. `https://huggingface.co/docs/peft/main/conceptual_guides/lora`, 2024.

[5] Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, et al. Olmo: Accelerating the science of language models. *arXiv preprint arXiv:2402.00838*, 2024.

[6] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[7] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[8] 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.

[9] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

[10] Disen Lan, Weigao Sun, Jiaxi Hu, Jusen Du, and Yu Cheng. Liger: Linearizing large language models to gated recurrent structures. *arXiv preprint arXiv:2503.01496*, 2025.

[11] Ben Mann, N Ryder, M Subbiah, J Kaplan, P Dhariwal, A Neelakantan, P Shyam, G Sastry, A Askell, S Agarwal, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 1:3, 2020.

[12] Sachin Mehta, Mohammad Hossein Sekhavat, Qingqing Cao, Maxwell Horton, Yanzi Jin, Chenfan Sun, Iman Mirzadeh, Mahyar Najibi, Dmitry Belenko, Peter Zatloukal, et al. Openelm: An efficient language model family with open-source training and inference framework. *arXiv e-prints*, pages arXiv–2404, 2024.

[13] Jean Mercat, Igor Vasiljevic, Sedrick Keh, Kushal Arora, Achal Dave, Adrien Gaidon, and Thomas Kollar. Linearizing large language models. *arXiv preprint arXiv:2405.06640*, 2024.

[14] William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. *arXiv preprint arXiv:2404.08819*, 2024.

[15] Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*, 101, 2024.

[16] Julien Siems, Timur Carstensen, Arber Zela, Frank Hutter, Massimiliano Pontil, and Riccardo Grazzi. Deltaproduct: Improving state-tracking in linear rnns via householder products. *arXiv preprint arXiv:2502.10297*, 2025.

[17] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html*, 3(6):7, 2023.

[18] 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.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[20] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[21] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *arXiv preprint arXiv:2406.06484*, 2024.

[22] Michael Zhang, Simran Arora, Rahul Chalamala, Alan Wu, Benjamin Spector, Aaryan Singhal, Krithik Ramesh, and Christopher Ré. Lolcats: On low-rank linearizing of large language models. *arXiv preprint arXiv:2410.10254*, 2024.