TPTT: Transforming Pretrained Transformers into Titans

Fabien Furfaro*

2025

Abstract

Recent advances in large language models (LLMs) have led to remarkable progress in natural language processing, but their computational and memory demands remain a significant challenge, particularly for long-context inference. We introduce TPTT (Transforming Pretrained Transformers into Titans), a novel framework for enhancing pretrained Transformers models with efficient linearized attention mechanisms and advanced memory management. TPTT employs techniques such as Memory as Gate (MaG) and mixed linearized attention (LiZA). It is fully compatible with the Hugging Face Transformers library, enabling seamless adaptation of any causal LLM through parameter-efficient fine-tuning (LoRA) without full retraining. We show the effectiveness of TPTT on the MMLU benchmark with models of approximately 1 billion parameters, observing substantial improvements in both efficiency and accuracy. For instance, Titans-Llama-3.2-1B achieves a 20% increase in Exact Match (EM) over its baseline. Statistical analyses and comparisons with recent state-of-the-art methods confirm the practical scalability and robustness of TPTT. The source code is available at `https://github.com/fabienfrfr/tptt` and the Python package at `https://pypi.org/project/tptt/`.

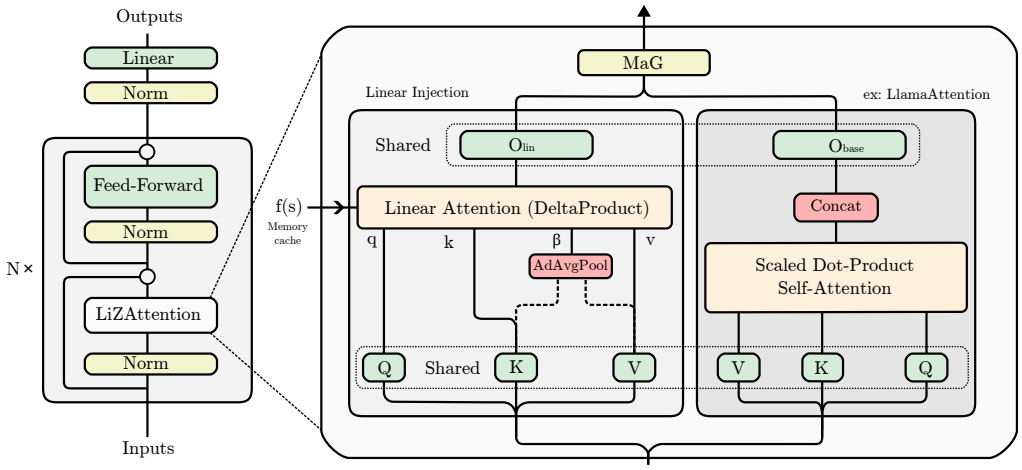

Figure 1: Overview of the TPTT architecture. On the left, the diagram illustrates a decoder-only architecture where linear attention is injected in parallel of vanilla attention (LiZAttention). On the right, the detailed architecture of the linearized attention mechanism is depicted, highlighting the 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

The success of transformer-based large language models (LLMs) [12, 21] has revolutionized natural language processing (NLP), enabling unprecedented progress across a wide range of tasks. However, the quadratic computational complexity and substantial memory requirements of the standard self-attention mechanism remain a significant bottleneck, particularly for long-context inference.

*fabien.furfaro@gmail.com

To address these challenges, recent research has explored several directions. Efficient attention mechanisms [10, 23] have been proposed to reduce the complexity of self-attention from quadratic to linear or near-linear, making it more tractable for long sequences. Recurrent architectures and internal memory mechanisms [3, 14] have also been developed to enhance the model's ability to capture long-range dependencies, drawing inspiration from cognitive memory in biological systems rather than hardware memory. Additionally, parameter-efficient fine-tuning approaches such as LoRA [9] allow for the adaptation of large pretrained models to new tasks without the need for full retraining.

2 Related Work

Despite these advances, most existing methods require significant architectural modifications or training from scratch, which limits their applicability to already pretrained models. For example, solutions like FlashAttention [4] and Mamba [7] focus on efficient architectures, while others such as LoLCat [24] and Liger [11] convert standard attention to linearized forms; notably, Liger exploits similar properties to those leveraged in this work, but does not rely on explicit linearization injection.

In this work, we introduce TPTT (Transforming Pretrained Transformers into Titans), a framework that transforms pretrained Transformers models into efficient and scalable architectures by incorporating linearized attention mechanisms and advanced internal memory augmentation. TPTT leverages techniques such as Memory as Gate (MaG) and mixed linearized attention (LiZA), drawing inspiration from the Titans architecture [3]. Our approach is fully compatible with existing frameworks and enables rapid adaptation of any causal LLM to long-context tasks via parameter-efficient fine-tuning with LoRA [9], without requiring full retraining. The goal is to unlock the potential of already trained models by equipping them with memory-augmented capabilities through lightweight adaptation.

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 [21]. To address this, linearized attention mechanisms approximate the softmax attention using linear operations, typically by projecting queries and keys through a feature map ϕ [10, 14, 22, 23]. This reduces computational and memory costs, enabling efficient processing of long sequences while maintaining modeling power [4, 7, 11, 24]. The output of softmax attention for an input sequence $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\} \in \mathbb{R}^{T \times D}$ is:

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

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

In linear attention, this is approximated as:

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

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 vector (or scalar) modulating the keys and values. The function ϕ denotes a feature mapping, such as ELU or kernel approximations [22]. In this work, β_i is constructed from average pooling of the keys and values.

3.2 Memory as Gate (MaG)

To further enhance long-range dependency modeling, we introduce an internal memory augmentation mechanism, Memory as Gate (MaG), inspired by the Titans architecture [3]. This mechanism enables the model to store and recall contextual information over extended sequences, analogous to cognitive memory (persistence, surprise, etc.). MaG combines the outputs of standard and linearized attention using a weighting parameter:

$$\mathbf{O} = \alpha \cdot \mathbf{O}_{\text{lin}} + (1 - \alpha) \cdot \mathbf{O}_{\text{base}} \tag{4}$$

where $\alpha \in [0, 1]$ is adapted during training. This allows the model to leverage both the efficiency of linear attention and the expressivity of softmax attention, and can be seen as a form of memory-augmented gating [3, 14, 16].

3.3 Parallel Delta Product Modeling

In this work, the feature mapping function of linear attention is based on the DeltaProduct operator, which generalizes the Delta rule to achieve higher expressivity while maintaining efficient parallelization. This approach addresses the limitations of linear recurrent architectures, which struggled to balance state-tracking power and hardware efficiency [18, 23].

The Delta rule [17], as used in associative memory models, updates the memory state sequentially as:

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \beta_t \left(\mathbf{v}_t - \mathbf{S}_{t-1}\mathbf{k}_t\right)\mathbf{k}_t^\top \qquad \text{(sequential, delta rule)} \tag{5}$$

where $\mathbf{S}_t$ is the memory state at time t, $\mathbf{k}_t$ and $\mathbf{v}_t$ are the key and value, and β_t is a gating parameter. This update is inherently sequential and thus not directly parallelizable.

DeltaNet: Parallelization via Householder Product DeltaNet [23] enables efficient parallelization by chunking the sequence and vectorizing the update within each chunk using a compact product of Householder matrices. For a chunk of size C, the intra-chunk parallel update (order $n = 1$) is:

$$\mathbf{S}_{[t,\,t+C-1]} = \mathbf{S}_{t-1} + \mathbf{K}^\top \left[\mathbf{T}^{-1}\left(\mathbf{V} \odot \boldsymbol{\beta} - (\mathbf{K} \odot \boldsymbol{\beta})\mathbf{S}_{t-1}\right)\right] \qquad \text{(parallel, intra-chunk, order 1)} \tag{6}$$

where $\mathbf{K} \in \mathbb{R}^{C\times d}$ and $\mathbf{V} \in \mathbb{R}^{C\times d}$ are the stacked keys and values in the chunk, $\boldsymbol{\beta} \in \mathbb{R}^{C\times 1}$ is the gating vector, $\odot$ denotes element-wise multiplication, and

$$\mathbf{T} = \mathbf{I} - \operatorname{tril}\left((\mathbf{K} \odot \boldsymbol{\beta})\mathbf{K}^\top, -1\right) \tag{7}$$

with $\operatorname{tril}(\cdot, -1)$ extracting the strictly lower triangular part. This chunkwise formulation enables efficient batched computation on modern hardware by removing sequential dependencies within each chunk [18, 23]. To further increase expressivity, a non-linearity ϕ (e.g., GELU or tanh) can be applied to the state update:

$$\mathbf{S}_t = \phi(\mathbf{S}_{t-1} + \beta_t \left(\mathbf{v}_t - \mathbf{S}_{t-1}\mathbf{k}_t\right)\mathbf{k}_t^\top) \tag{8}$$

DeltaProduct: Generalization to Higher Order DeltaProduct [18] extends this approach by applying n rank-1 corrections per token, corresponding to a product of n Householder-like transformations. For each position i, the state update is defined recursively as:

$$H_{i,j} = H_{i,j-1} - \beta_{i,j}(H_{i,j-1}\mathbf{k}_{i,j} - \mathbf{v}_{i,j})\mathbf{k}_{i,j}^\top \qquad \text{for } j = 1, \ldots, n \tag{9}$$

where $H_{i,0} = H_{i-1}$ and $H_{i,n} = H_i$. This can be unrolled as:

$$H_i = A(\mathbf{x}_i)H_{i-1} + B(\mathbf{x}_i) \tag{10}$$

$$A(\mathbf{x}_i) = \prod_{j=1}^{n}\left(\mathbf{I} - \beta_{i,j}\mathbf{k}_{i,j}\mathbf{k}_{i,j}^\top\right) \tag{11}$$

$$B(\mathbf{x}_i) = \sum_{j=1}^{n}\left(\prod_{k=j+1}^{n}\left(\mathbf{I} - \beta_{i,k}\mathbf{k}_{i,k}\mathbf{k}_{i,k}^\top\right)\right)\beta_{i,j}\mathbf{k}_{i,j}\mathbf{v}_{i,j}^\top \tag{12}$$

where $\mathbf{k}_{i,j}$, $\mathbf{v}_{i,j}$, and $\beta_{i,j}$ are the key, value, and gate for the j-th virtual token at position i.

To construct the n virtual tokens per real token, we use a binomial finite difference expansion (derivative trick), as follows:

$$\mathbf{x}_t^{(m)} = \sum_{k=0}^{m}(-1)^k\binom{m}{k}\mathbf{x}_{t-k} \qquad \text{for } m = 0, \ldots, n-1 \tag{13}$$

where $\mathbf{x}_t^{(m)}$ is the m-th discrete derivative (virtual token) at position t [2].

The corresponding chunkwise parallel update is:

$$\mathbf{S}_{[t,\,t+C-1]} = \mathbf{S}_{t-1} + \mathbf{K}_h^\top \left[\mathbf{T}_h^{-1}\left(\mathbf{V}_h \odot \boldsymbol{\beta}_h - (\mathbf{K}_h \odot \boldsymbol{\beta}_h)\mathbf{S}_{t-1}\right)\right] \qquad \text{(parallel, intra-chunk, order } n\text{)} \tag{14}$$

where, for each Householder step h, $\mathbf{K}_h, \mathbf{V}_h \in \mathbb{R}^{(nC)\times d}$ and $\boldsymbol{\beta}_h \in \mathbb{R}^{(nC)\times 1}$ are the keys, values, and gates for the h-th virtual token within the chunk, and

$$\mathbf{T}_h = \mathbf{I} - \operatorname{tril}\left((\mathbf{K}_h \odot \boldsymbol{\beta}_h)\mathbf{K}_h^\top, -1\right). \tag{15}$$

This matches our implementation: the input is expanded into n virtual tokens per real token using the derivative trick, and all $\mathbf{T}_h$ are inverted in parallel for efficient batched computation on GPU. The final state $\mathbf{S}_{t+C-1}$ initializes the next chunk.

Expressivity and Theoretical Properties. 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) [15, 18]. 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 [9], 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 [10].

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_{lin} using a feature map ϕ:
$o_{\text{lin}}[t] = \frac{\sum_{i=1}^{t} \phi(q_t)^\top \phi(k_i) v_i}{\sum_{i=1}^{t} \phi(q_t)^\top \phi(k_i)}$
Store intermediate states in a memory cache for recurrent information.
4: **Vanilla (Softmax) Attention:**
Compute standard self-attention output o_{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 α:
$o = \alpha \cdot o_{\text{lin}} + (1 - \alpha) \cdot o_{\text{base}}$
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 [10]. 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) [5, 9], 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 [5, 9]. 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 [5].

4.2 Dynamic Memory as Gate Scheduling (LiZA MaG Callback)

A important component of the training process is the LiZA MaG 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 [8] as the primary evaluation suite. Training was conducted on 500 samples from the `yahma/alpaca-cleaned` dataset [19] 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	[20]
Titans-OpenELM-1_1B	1.3188	1651.1	1.51	0.254	5.85e18	0.704	[13]
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	[6]

Table 1: Training performance metrics for TPTT models.

5.3 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 [8].

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.

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

Model	EM ± Std	PEM ± Std	PQEM ± 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.

5.4 Discussion and Comparison

Compared to recent state-of-the-art methods such as Mamba [7], LoLCat [24], and Liger [11], 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) [9] 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 [8] shows significant improvements in both efficiency and accuracy, with robust statistical analyses and favorable comparisons to state-of-the-art methods [7, 11, 24].

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 [3], 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, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. It's all connected: A journey through test-time memorization, attentional bias, retention, and online optimization. *arXiv preprint arXiv:2504.13173*, 2025.

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

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

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

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.