TPTT : Transformation de Transformers Pré-entraînés en Titans

Fabien Furfaro*

2025

Abstract

Les avancées récentes dans les grands modèles de langage (LLMs) ont permis des progrès remarquables en traitement automatique du langage naturel, mais leurs demandes computationnelles et mémorielles restent un défi majeur, en particulier pour l'inférence en contexte long. Nous présentons TPTT (Transforming Pretrained Transformer into Titans), un nouveau cadre pour améliorer les modèles Transformer pré-entraînés avec des mécanismes d'attention linéarisés efficaces et une gestion avancée de la mémoire. TPTT tire parti de techniques telles que la Mémoire comme Porte (MaG) et l'attention linéarisée mixte (LiZA), et est entièrement compatible avec la bibliothèque Hugging Face Transformers, permettant une adaptation transparente de tout LLM causal via un ajustement fin paramétrique efficace (LoRA) sans réentraînement complet. Nous démontrons l'efficacité de TPTT sur le benchmark MMLU avec des modèles d'environ 1 milliard de paramètres, observant des améliorations substantielles en termes d'efficacité et de précision. Par exemple, Titans-Llama-3.2-1B atteint une augmentation de 20% en Exact Match (EM) par rapport à sa référence. Des analyses statistiques détaillées et des comparaisons avec les méthodes récentes de l'état de l'art confirment la scalabilité pratique et la robustesse de TPTT. Le code source est disponible à l'adresse `https://github.com/fabienfrfr/tptt`.

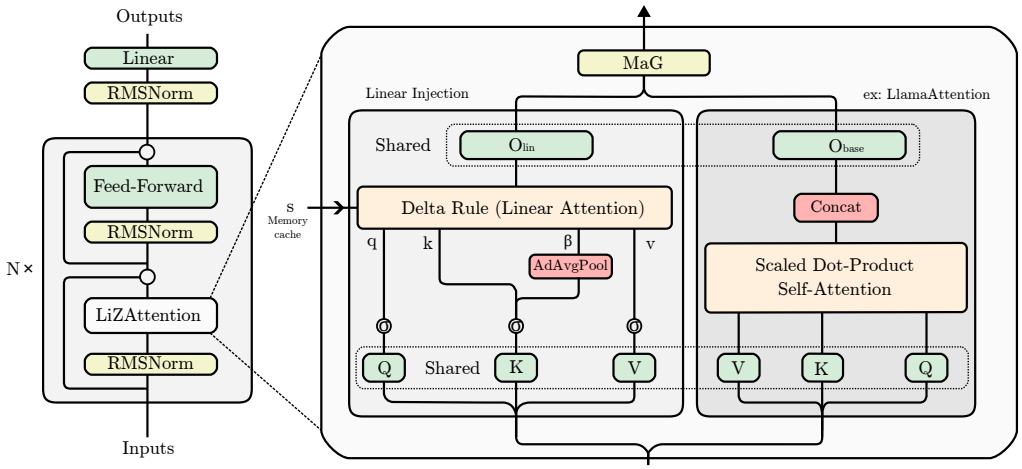

Figure 1: Aperçu de l'architecture du cadre TPTT. À gauche, le diagramme illustre une architecture de décodeur où l'attention linéaire est injectée en parallèle de l'attention vanille (LiZAttention). À droite, l'architecture détaillée du mécanisme d'attention linéarisée est représentée, mettant en évidence les poids partagés pour les projections de requête (Q), de clé (K) et de valeur (V). Il montre également la gestion de la mémoire d'état (S) et la combinaison des sorties par le mécanisme de pondération Memory as Gate (MaG). Le diagramme met l'accent sur l'intégration des mécanismes d'attention linéarisée et des techniques avancées de gestion de la mémoire, telles que Delta Rule et AdaptativeAvgPool1D, contribuant au traitement et à la génération de sorties.

1 Introduction

Le succès des grands modèles de langage basés sur les transformeurs (LLMs) [11,16] a révolutionné le traitement automatique du langage naturel (NLP), permettant des progrès sans précédent dans une large gamme de tâches. Cependant, la complexité computationnelle quadratique et les exigences substantielles en mémoire du mécanisme

*fabien.furfaro@gmail.com

d'auto-attention standard restent un goulot d'étranglement significatif, en particulier pour l'inférence en contexte long.

Pour relever ces défis, des recherches récentes ont exploré plusieurs directions. Des mécanismes d'attention efficaces [9, 18] ont été proposés pour réduire la complexité de l'auto-attention de quadratique à linéaire ou quasi-linéaire, la rendant plus tractable pour les longues séquences. Des architectures récurrentes et des mécanismes de mémoire interne [2, 13] ont également été développés pour améliorer la capacité du modèle à capturer les dépendances à long terme, s'inspirant de la mémoire cognitive dans les systèmes biologiques plutôt que de la mémoire matérielle. De plus, des approches d'ajustement fin paramétrique efficaces telles que LoRA [8] permettent l'adaptation de grands modèles pré-entraînés à de nouvelles tâches sans avoir besoin d'un réentraînement complet.

2 Travaux Connexes

Malgré ces avancées, la plupart des méthodes existantes nécessitent des modifications architecturales significatives ou un entraînement à partir de zéro, ce qui limite leur applicabilité aux modèles déjà pré-entraînés. Par exemple, des solutions comme FlashAttention [3] et Mamba [6] se concentrent sur des architectures efficaces, tandis que d'autres comme LoLCat [19] et Liger [10] convertissent l'attention standard en formes linéarisées ; notamment, Liger exploite des propriétés similaires à celles utilisées dans ce travail, mais ne repose pas sur une injection de linéarisation explicite.

Dans ce travail, nous introduisons TPTT (Transforming Pretrained Transformer into Titans), un cadre qui transforme les modèles de transformateurs pré-entraînés en architectures plus efficaces et scalables en incorporant des mécanismes d'attention linéarisés et une augmentation avancée de la mémoire interne. TPTT tire parti de techniques telles que la Mémoire comme Porte (MaG) et l'attention linéarisée mixte (LiZA), s'inspirant de l'architecture Titans [2]. Notre approche est entièrement compatible avec les cadres existants et permet une adaptation rapide de tout LLM causal à des tâches à long contexte via un ajustement fin paramétrique efficace avec LoRA [8], sans nécessiter de réentraînement complet. L'objectif est de débloquer le potentiel des modèles déjà entraînés en les équipant de capacités augmentées par la mémoire grâce à une adaptation légère.

3 Methodologie

3.1 Mécanismes d'Attention Linéarisés

L'auto-attention standard dans les transformeurs calcule les interactions par paires entre tous les jetons, ce qui entraîne une complexité quadratique par rapport à la longueur de la séquence [16]. Pour remédier à cela, les mécanismes d'attention linéarisés approximient l'attention softmax en utilisant des opérations linéaires, généralement en projetant les requêtes et les clés à travers une carte de caractéristiques ϕ [9, 13, 17, 18]. Cela réduit les coûts computationnels et mémoriels, permettant un traitement efficace des longues séquences tout en maintenant la puissance de modélisation [3, 6, 10, 19]. La sortie de l'attention softmax pour une séquence d'entrée $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\} \in \mathbb{R}^{T \times D}$ est :

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

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

En attention linéaire, cela est approximé comme suit :

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

où ϕ est une fonction de mappage de caractéristiques, telle que ELU ou des approximations de noyau [17].

3.2 Mémoire comme Porte (MaG)

Pour améliorer davantage la modélisation des dépendances à long terme, nous introduisons un mécanisme d'augmentation de la mémoire interne, Mémoire comme Porte (MaG), inspiré de l'architecture Titans [2]. Contrairement à la mémoire matérielle, ce mécanisme permet au modèle de stocker et de rappeler des informations contextuelles sur des séquences étendues, de manière analogue à la mémoire cognitive. MaG combine les sorties de l'attention standard et linéarisée en utilisant un paramètre de pondération :

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

où $\alpha \in [0, 1]$ est adapté pendant l'entraînement. Cela permet au modèle de tirer parti à la fois de l'efficacité de l'attention linéaire et de l'expressivité de l'attention softmax, et peut être vu comme une forme de gating augmentée par la mémoire [2, 13].

3.3 Modélisation Parallèle de la Règle Delta

Dans ce travail, la fonction de mappage des caractéristiques de l'attention linéaire est approximée par DeltaNet [18]. La mise à jour récurrente des états de mémoire interne est formulée de deux manières, suivant :

- **Forme Fermée (intra-chunk) :**

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \sum_{i=1}^{C} \mathbf{v}_i \mathbf{k}_i^\top \tag{5}$$

  où C est la taille du chunk et $\mathbf{S}_t$ est l'état mis à jour.

- **Forme Récurrente (inter-chunk) :**

$$\mathbf{S}_{t+1} = \mathbf{S}_t + \mathbf{v}_{t+1} \mathbf{k}_{t+1}^\top \tag{6}$$

  L'état final d'un chunk devient l'état initial du suivant, soutenant une utilisation efficace de la mémoire pour les longues séquences.

La sortie $\mathbf{O}_{\text{lin}}$ est ensuite calculée à partir de ces états de mémoire.

3.4 Intégration avec les Modèles Pré-entraînés

Notre approche injecte des modules d'attention linéarisés et d'augmentation de mémoire dans les modèles de transformateurs pré-entraînés. Le processus implique :

1. **Identification des Modules Cibles :** Les couches d'attention clés à modifier sont identifiées à l'aide d'outils tels que `get_tptt_model`.
2. **Modification des Couches d'Attention :** Ces couches sont remplacées ou étendues avec le module proposé `LiZAttention`, qui implémente à la fois l'attention linéaire et softmax avec un partage de poids de projection linéaire et MaG.
3. **Entraînement et Ajustement Fin :** Le modèle modifié est ajusté finement en utilisant des méthodes paramétriques efficaces telles que LoRA [8], assurant une adaptation optimale aux nouveaux mécanismes sans réentraînement complet.

Cette procédure permet la transformation de tout LLM causal pré-entraîné en une architecture efficace augmentée de mémoire avec un minimum de surcharge, et ce sans aucune nouvelle couche.

3.5 Module LiZAttention

Le module `LiZAttention` est un composant central de l'architecture TPTT, conçu pour combiner de manière synergique les mécanismes d'attention linéarisée et standard (softmax). Cette approche hybride tire parti de l'efficacité computationnelle de l'attention linéaire tout en conservant l'expressivité de l'attention vanille. Pour soutenir l'inférence en contexte long, `LiZAttention` maintient un cache d'états intermédiaires et implémente un mécanisme d'information récurrente pour une gestion efficace de la mémoire interne [9].

Algorithm 1 Passe Avant de LiZAttention

Require: hidden_states $\in \mathbb{R}^{B\times L\times D}$ ▷ Taille du lot B, longueur de la séquence L, dimension d'intégration D
Require: mask ▷ Masque d'attention pour le padding/la causalité

1: **Projection :**
Calculer les requêtes q, les clés k, les valeurs v via des projections apprises :
$q \in \mathbb{R}^{B\times H\times L\times d_h}$, $k, v \in \mathbb{R}^{B\times H_k\times L\times d_h}$

2: **Appliquer le Masque d'Attention :**
Appliquer mask à k et v pour gérer le padding et restreindre l'attention.

3: **Attention Linéaire :**
Calculer la sortie de l'attention linéaire o_{lin} en utilisant une carte de caractéristiques ϕ :
$o_{\text{lin}}[t] = \frac{\sum_{i=1}^{t} \phi(q_t)^\top \phi(k_i) v_i}{\sum_{i=1}^{t} \phi(q_t)^\top \phi(k_i)}$
Stocker les états intermédiaires dans un cache mémoire pour l'information récurrente.

4: **Attention Vanille (Softmax) :**
Calculer la sortie de l'auto-attention standard o_{base} (optionnellement tronquée pour les longues séquences) :
$o_{\text{base}} = \text{Softmax}\left(\frac{qk^\top}{\sqrt{d_h}}\right) v$

5: **Combiner les Sorties (Mémoire comme Porte) :**
Calculer la sortie finale en utilisant un paramètre de porte apprise 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 Gestion Efficace de la Mémoire Interne

Le cache d'états intermédiaires maintenu par `LiZAttention` permet une information récurrente, soutenant efficacement l'inférence en contexte long sans surcharge computationnelle excessive [9]. Cette approche permet au modèle de passer à l'échelle pour des séquences plus longues, en tirant parti à la fois du contexte local et global.

4 Procédure d'Entraînement

4.1 Ajustement Fin Efficace des Paramètres avec LoRA

Pour adapter l'architecture TPTT aux tâches en aval, nous employons l'Adaptation de Faible Rang (LoRA) [4,8], une technique d'ajustement fin efficace des paramètres qui injecte des matrices de faible rang entraînables dans des couches de projection sélectionnées tout en gelant les poids du modèle d'origine. Cette approche réduit considérablement le nombre de paramètres entraînables et les exigences en mémoire, tout en maintenant des performances comparables à un ajustement fin complet [4, 8]. Pour TPTT, LoRA est configuré avec un rang de 8, $\alpha = 16$, et un taux de dropout de 0,05. L'ajustement fin cible les modules de projection principaux, spécifiquement `[q_proj, k_proj, v_proj, o_proj]` pour Llama/Mistral et `[qkv_proj, out_proj]` pour OpenELM [4].

4.2 Planification Dynamique de la Mémoire comme Porte (Callback LiZA MaG)

Un composant critique du processus d'entraînement est le callback LiZA MaG, qui ajuste dynamiquement le paramètre de pondération Mémoire comme Porte (MaG) pendant l'entraînement. Le poids MaG est initialisé à 0,01 et augmenté linéairement à 0,5 sur les 100 premières étapes, facilitant une transition en douceur de la dépendance à l'attention vanille (softmax) vers l'attention linéarisée. Cet horaire permet au modèle d'équilibrer efficacement les deux mécanismes d'attention, optimisant les performances tout au long de l'entraînement. Le callback est intégré directement dans la boucle d'entraînement, assurant un contrôle adaptatif du paramètre MaG et améliorant l'adaptabilité et l'efficacité du modèle.

5 Expériences et Résultats

5.1 Configuration Expérimentale

Nous avons évalué le cadre TPTT sur plusieurs modèles de langage pré-entraînés avec environ 1 milliard de paramètres, en utilisant le benchmark MMLU [7] comme suite d'évaluation principale. L'entraînement a été effectué sur 500 échantillons du jeu de données `yahma/alpaca-cleaned` [14] pendant 5 époques, avec une longueur de séquence maximale de 384 jetons, une taille de lot de 3, et un taux d'apprentissage de 5×10^{-4}. La

formation mixte de précision et le clippage des gradients à 1,0 ont été employés pour optimiser l'efficacité et la stabilité computationnelles. Toutes les expériences ont été réalisées sur des GPU NVIDIA Tesla T4 (plateforme Kaggle). Les modèles entraînés et les métriques détaillées sont publics sur le Hugging Face Model Hub[1], avec des journaux d'entraînement complets accessibles via Hugging Face TensorBoard.

5.2 Résultats de l'Entraînement

Le tableau 1 résume les métriques de performance de l'entraînement pour divers modèles TPTT. Les résultats indiquent une apprentissage constant et efficace à travers les architectures, comme en témoignent les faibles valeurs de perte finale et les normes de gradient stables. L'utilisation de l'Adaptation de Faible Rang (LoRA) et du mécanisme Mémoire comme Porte (MaG) s'est avérée efficace pour optimiser la dynamique et la convergence de l'entraînement.

Modèle	Perte	Temps d'Entraînement (s)	Échantillons/s	Étapes/s	FLOPs Totaux	Norme du Gradient	Références Basées Sur
Titans-Llama-3.2-1B	1,375	1654,1	1,51	0,254	5,62e18	2,68	[15]
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: Métriques de performance d'entraînement pour les modèles TPTT.

5.3 Métriques d'Évaluation et Résultats du Benchmark

Pour l'évaluation, nous nous concentrons sur les métriques standard dans le benchmarking LLM et QA : Exact Match (EM), Partial Exact Match (PEM), et Partial Quasi Exact Match (PQEM). Ces métriques mesurent respectivement la correction stricte, le chevauchement partiel, et la quasi-exactitude entre les sorties du modèle et les réponses de vérité terrain, fournissant une vue nuancée de la performance du modèle [7].

Le tableau 2 présente les résultats du benchmark MMLU dans le cadre one-shot. Les modèles TPTT, et en particulier Titans-Llama-3.2-1B, surpassent systématiquement leurs homologues de base en EM, avec des scores PEM et PQEM compétitifs. Cela démontre l'avantage de l'intégration de l'attention linéarisée et des mécanismes de mémoire interne pour des tâches complexes de compréhension du langage.

Modèle	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: Résultats du benchmark MMLU (one-shot) avec analyse statistique. Chaque paire regroupe un modèle Titans et son homologue de base.

5.4 Discussion et Comparaison

Comparé aux méthodes récentes de l'état de l'art telles que Mamba [6], LoLCat [19], et Liger [10], TPTT se distingue en permettant la transformation de modèles pré-entraînés existants sans réentraînement complet, tout en maintenant de solides performances de benchmark. Les améliorations observées en EM et les scores compétitifs en PEM/PQEM soulignent l'efficacité de l'attention linéarisée et de l'augmentation de mémoire de TPTT pour une adaptation efficace et robuste des LLM. Ces résultats confirment que TPTT est une solution pratique et scalable pour améliorer les LLM pré-entraînés, en particulier dans des environnements contraints en ressources.

6 Discussion et Conclusion

Dans cet article, nous avons introduit TPTT, un nouveau cadre pour améliorer les modèles Transformer pré-entraînés en intégrant des mécanismes d'attention linéarisés efficaces et une augmentation de la mémoire interne.

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

Notre approche utilise l'ajustement fin paramétrique efficace (LoRA) [8] pour permettre l'adaptation rapide des grands modèles de langage (LLMs) à des tâches en contexte long, sans avoir besoin d'un réentraînement complet. Les résultats expérimentaux sur le benchmark MMLU [7] démontrent des améliorations significatives en termes d'efficacité et de précision, avec des analyses statistiques robustes et des comparaisons favorables aux méthodes de pointe [6, 10, 19].

Implications Pratiques. TPTT offre une solution scalable et pratique pour déployer des LLMs haute performance dans des environnements contraints en ressources. L'intégration de mécanismes d'attention linéarisés et d'augmentation de mémoire réduit les exigences computationnelles et mémorielles, rendant les modèles de langage avancés plus accessibles pour des applications réelles. L'utilisation de LoRA permet un ajustement fin efficace et flexible, permettant une adaptation rapide à de nouvelles tâches et domaines.

Limitations. Notre évaluation actuelle est limitée à des modèles de taille modérée (environ 1 milliard de paramètres). L'échelle de TPTT à des architectures plus grandes et à des tâches plus diversifiées peut introduire de nouveaux défis, y compris une complexité d'ajustement accrue et la nécessité d'une optimisation supplémentaire des mécanismes de mémoire. Bien que nos résultats soient prometteurs, une validation plus large sur des benchmarks supplémentaires et des scénarios réels est nécessaire pour évaluer pleinement la généralisabilité et la robustesse de l'approche.

Directions Futures. Les travaux futurs se concentreront sur l'optimisation du processus d'intégration, l'exploration de mécanismes de mémoire interne plus sophistiqués [2], et l'extension de l'évaluation à des modèles plus grands et à une gamme plus large de benchmarks. Des recherches supplémentaires étudieront les approches hybrides et l'interaction entre l'attention linéarisée, l'augmentation de mémoire et d'autres techniques axées sur l'efficacité.

En résumé, TPTT fournit un cadre pratique, scalable et efficace pour la mise à niveau des Transformers préentraînés, avec des résultats empiriques solides et des implications prometteuses pour l'avenir de la modélisation du langage efficace.

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] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.