Exploring Efficient Methods for Transformer-based Foundation Language Models

Huiyin Xue

Doctor of Philosophy
School of Computer Science
University of Sheffield

September 2025

Abstract

Transformer-based language models have achieved huge success in various natural language processing tasks, but their efficiency remains a critical challenge. The core limitations stem from the computationally intensive dot-product attention mechanism, which scales quadratically with input length, and the increasing memory footprint associated with large model sizes and extensive vocabularies. These issues hinder their deployment on resource-constrained devices and limit their ability to process long contexts.

This thesis explores novel methods to enhance the efficiency of Transformer-based models, focusing on architectural modifications during the pre-training stage. It is presented as a collection of three peer-reviewed publications, each addressing a specific component of the Transformer architecture. The first work introduces a parameter-efficient embedding layer that uses a hashing function to support an unlimited vocabulary with a fixed-size embedding matrix. This approach effectively breaks the rigid, one-to-one mapping between tokens and embeddings, significantly reducing memory consumption. The second work tackles the parameter redundancy within the multi-head attention mechanism. It proposes a more efficient alternative that uses a single shared projection matrix and multiple head embeddings, substantially reducing the number of attention-related parameters while preserving model performance. The final work systematically analyses the key design principles of the dot-product attention mechanism. The findings provide insights into what makes attention so effective and offer a foundation for developing more streamlined and efficient attention mechanisms in the future.

This thesis demonstrates that fine-grained architectural modifications during pre-training can yield substantial improvements in model efficiency. The proposed methods are orthogonal to existing post-training compression techniques, providing a complementary approach to creating more scalable and deployable language models. The findings collectively contribute to a deeper understanding of the core components of Transformer models and pave the way for designing next-generation language models that are both powerful and efficient.

Acknowledgements

Completing this thesis would not have been possible without the guidance and support of many people.

First, I want to thank my brilliant, erudite, extraordinary supervisors - Professor Nikolaos Aletras and Dr. Nafise Sadat Moosavi. My primary supervisor, Professor Nikolaos Aletras, deserves my deepest thanks. His unreserved support and exceptional professional skills were a constant source of confidence and direction. His guidance helped me not only to complete this research but also to grow as a researcher, a real human. I am also profoundly grateful for Dr. Nafise Sadat Moosavi. Her rigorous and thoughtful thinking pushed me to refine my arguments and ensure the integrity of my work. Her high standards were a true motivation throughout this process in the past two years.

I also wish to express my sincere appreciation to Dr. Milan Gritta, my mentor during my internship at the Huawei London Research and Development Center. His unwavering encouragement to not give up when facing frustrating bugs and negative results made a significant impact on my resilience as a researcher. His guidance went beyond the project itself, teaching me a valuable lesson in perseverance that I will carry forward.

Second, I would also like to thank those who share their knowledge with me and inspire my research. Some interesting papers recommended by Miles Williams enriched my knowledge and greatly inspired me to find a neat solution quickly to the problem I encountered in the internship. Atsuki Yamaguchi's work on pretraining objectives for encoder-only language models provided me a good tutorial in the early stage of my PhD. Professor Tiefeng Ma in SWUFE planted a seed on applying analogy between different disciplines in my heart.

Third, thanks to sisters and brothers who provided a warm learning atmosphere. Thanks to Cass, Danae, Yue, Miles, Atsuki, Xingwei, Constantinos, Mali, Yida, Ahmed, Sam, George, Mingzi, Yanwen, Vynska, Anthony, Peter, Katerina, Makis, Milan, Fenia, Victor, Ronald, Dimitris, Guchun, Favour, Jasivan, Maggie, Reem, Owen, Ben, Varvara etc.

Thanks to the generous university and the Research Software Engineering team, providing hardware to train language models.

Thanks to my previous guide in my basic education: former coach in Physics Olympiad, Guangmin Hong; homeroom teacher in high school, Zhong Lyu. They always encourage me to actively self-study and tackle challenging problems.

Thanks to my friends and teammates in SWUFE, they left me many wonderful memories which encouraged me to stick on interest while pursuing a PhD.

Finally, thanks to my parents for their unreserved support. They made great efforts to pre-train

their ‘human-shaped baby language model’. And thanks to myself, who completed the value alignment.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Huiyin Xue*)

To you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Transformer-based language models (Singh, 2025; Liu et al., 2024a; Yang et al., 2024a; Vaswani et al., 2017) have obtained great success on various down-stream tasks in recent years. The classic core components in them include: (1) the embedding layers, (2) stacked transformer layers, where each one does token mixing and channel mixing sequentially leveraging a feed-forward module stacked above an attention module; (3) task specific classifier.

When actively applied, efficiency becomes a crucial demand for models. The success of Transformer-based language models stems from their dot-product attention mechanism. However, this mechanism is computationally inefficient because it requires the model to calculate pairwise similarities between every token in a sequence to determine their relative importance. This inefficient process limits the model's ability to handle long inputs. Furthermore, while the trend of scaling model parameters has improved their capabilities, it also significantly increases inference latency and memory consumption. This makes it difficult to deploy these large models on devices with limited resources, such as smartphones or other edge devices.

Besides hardware-specific optimisations such as IO-aware flash attention and paged attention, researchers have developed different methods to optimise the efficiency of large language models across various stages of their life cycle:

- In pretraining, three main lines of work exist: model distillation, low-bit pretraining, and efficient architectures. Model distillation (Gu et al., 2024; Hsieh et al., 2023) leverages a larger, more complex model to transfer its knowledge to a smaller, more efficient one by aligning their outputs. Low-bit pretraining (Nielsen et al., 2025; Ouyang et al., 2025) reduces the bit-width used for computation, adopting low-precision values for training to decrease the memory footprint and accelerate the process. Most research on efficient architectures focuses on developing new token mixers with lower computational complexity to replace the standard dot-product attention module (Yang et al., 2025b; He

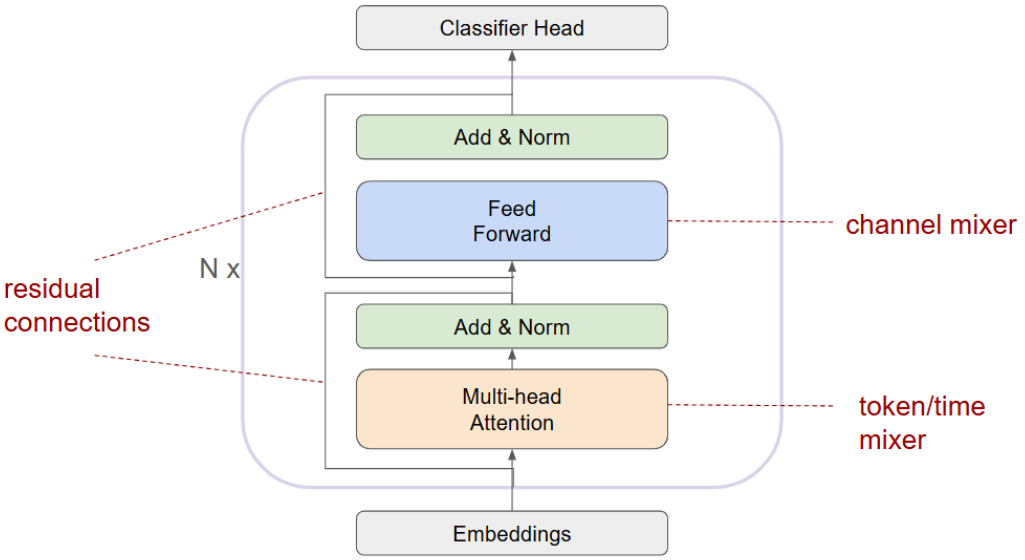


Figure 1.1: Illustration of the architecture of Transformer-based language models

et al., 2025; Lin et al., 2025; Siems et al., 2025; Peng et al., 2025; Yang et al., 2024b; Dao and Gu, 2024; Yang et al., 2024b; Qin et al., 2024; Peng et al., 2024; Gu and Dao, 2023; Poli et al., 2023; Peng et al., 2023; Orvieto et al., 2023; Gu et al., 2022; Lee-Thorp et al., 2022; Wang et al., 2020). The aim is to reduce training time and inference latency, especially with long contexts. Some work also focuses on alleviating parameter redundancy in specific components, such as the embedding layer and the attention module.

- Post-training (the stage after pretraining) optimisation methods for efficient Transformer-based language models fall into two categories: general compression technologies and architecture-dependent approaches.
  - General compression techniques include pruning, quantisation, and layer-skip, and can be applied to various neural network architectures:
    - **Pruning** (Dettmers et al., 2024; Sun et al., 2024; Frantar and Alistarh, 2023) reduces the size of a trained network by identifying and removing redundant parameters;
    - **Quantisation** (Dettmers et al., 2024; Frantar et al., 2023; Xiao et al., 2023; Dettmers et al., 2022) improves memory efficiency and speeds up computation by converting high-precision floating-point values into lower-precision integers;
    - **Layer-skip** (Gromov et al., 2025; Varshney et al., 2024; Bae et al., 2023; Schuster et al., 2021) optimises the forward pass by either allowing tokens to exit the network early or by identifying and removing redundant layers.

- Architecture-Dependent approaches are tailored to specific network architectures and often focus on addressing latency issues, particularly those related to the quadratic time complexity of standard dot-product computation:
  - **Cross-architecture fine-tuning** (Mercat et al., 2024; Bick et al., 2024; Choi, 2024; Mao, 2022; Kasai et al., 2021) involves transferring parameters from a large, pre-trained language model to a more efficient target model (like a recurrent-style linear attention). The new model is then fine-tuned over fewer steps.
  - **KV eviction** (Chen et al., 2024; Liu et al., 2024b; Devoto et al., 2024; Zhang et al., 2023; Liu et al., 2023) reduces attention computation during inference by removing redundant cached key-value pairs;
  - **Speculative decoding** (Gritta et al., 2025; Zimmer et al., 2025; Miao et al., 2024; He et al., 2024; Cai et al., 2024; Li et al., 2024b; Ankner et al., 2024; Mamou et al., 2024) maximises the parallelism of auto-regressive language model inference to decrease latency. It uses a smaller, efficient model to draft multiple tokens, which are then verified in parallel by a larger, more powerful model.

This thesis focuses on efficient architectures for the pre-training stage. These approaches are orthogonal to general compression methods, which can be applied afterwards. Pre-training language models from scratch offers greater flexibility to modify fine-grained components within the model. This also allows us to isolate the effects of the architecture itself from the catastrophic forgetting of knowledge stored in existing pre-trained language models.

1.1 Research Aims and Objectives

The central aim of this thesis is to explore the reducibility of the classic components in prevalent Transformer-based foundation language models. It is expanded along the following dimensions for selecting the target components to investigate:

Compressibility of inelastic components Transformer-based pre-trained language models generally map each token to its corresponding embedding, which results in a rigid, one-to-one mapping. This approach creates inelastic embedding matrices whose size grows linearly with the pre-defined vocabulary size. A key focus is on making the embedding layer in encoder-only Transformer models more compressible by breaking this one-to-one mapping. The goal is to free future models from relying on large vocabularies and over-segmented tokenisation, allowing them to support an unlimited vocabulary of all possible tokens in a corpus.

Parameter redundancy between multiple subspaces Transformer-based language models use multiple "attention heads", which are sets of linear projections, to process the same query, key, and value pairs. Each head operates on a different subspace to create a richer, more accurate representation of the input data. The goal of this research is to show that there is substantially large parameter redundancy when generating these multiple subspaces for the attention heads. By addressing this redundancy, we can reduce the model's memory footprint, decrease computational time, and shrink the size of the key-value cache in transformer-based pre-trained language models.

Design principles for the most frequently iterated components The token mixer, as known as the attention mechanism, is one of the most frequently refined components in Transformer-based language models. While previous research has used ideas from graph neural networks and recurrent neural networks to reduce the computational complexity of standard dot-product attention, there has been limited exploration of its core design principles. By deconstructing and examining these principles, we can gain a deeper understanding of what makes attention so effective for language modeling. This knowledge could then lead to new ways of simplifying language models while maintaining their performance.

1.2 Thesis Overview: Publications and Contributions

This thesis presents a collection of three first-author, peer-reviewed publications that focus on improving the efficiency of Transformer-based language models. Each chapter explores a distinct component of the original Transformer architecture—the embedding layer, the multi-head attention mechanism, and the dot-product attention form—to identify and optimise areas for greater efficiency.

Chapter 2, based on the paper "HashFormers: Towards Vocabulary-independent Pre-trained Transformers" (EMNLP 2022) (Xue and Aletras, 2022), tackles the memory footprint of the embedding layer. It introduces HASHFORMERS, a novel family of parameter-efficient, encoder-only Transformer models. By using a hashing function, HASHFORMERS can support an unlimited vocabulary with a significantly smaller fixed-sized embedding matrix, making the model more memory efficient.

Drawn from the publication "Pit One Against Many: Leveraging Attention-head Embeddings for Parameter-efficient Multi-head Attention" (EMNLP 2023 Findings) (Xue and Aletras, 2023), Chapter 3 addresses the parameter inefficiency of the multi-head attention mechanism. It introduces a more parameter-efficient alternative that replaces the traditional head-wise projection matrices with a single shared projection matrix and multiple head embeddings (MHE). This approach drastically reduces the number of parameters needed for attention while

maintaining performance.

Chapter 4 based on the paper "Deconstructing Attention: Investigating Design Principles for Effective Language Modeling" (Under Review at ACL Rolling Review, July 2025 cycle), investigates the standard dot-product attention mechanism. It systematically deconstructs this component to identify and analyse the fundamental design principles that contribute to effective language modeling. The insights gained from this analysis could pave the way for future, more efficient attention mechanisms.

Finally, Chapter 5 synthesises the key findings from these three empirical studies and outlines promising directions for future research. The overall goal is to continue exploring and developing efficient methods for designing and optimising Transformer-based foundation language models.

Chapter 2

Publication I

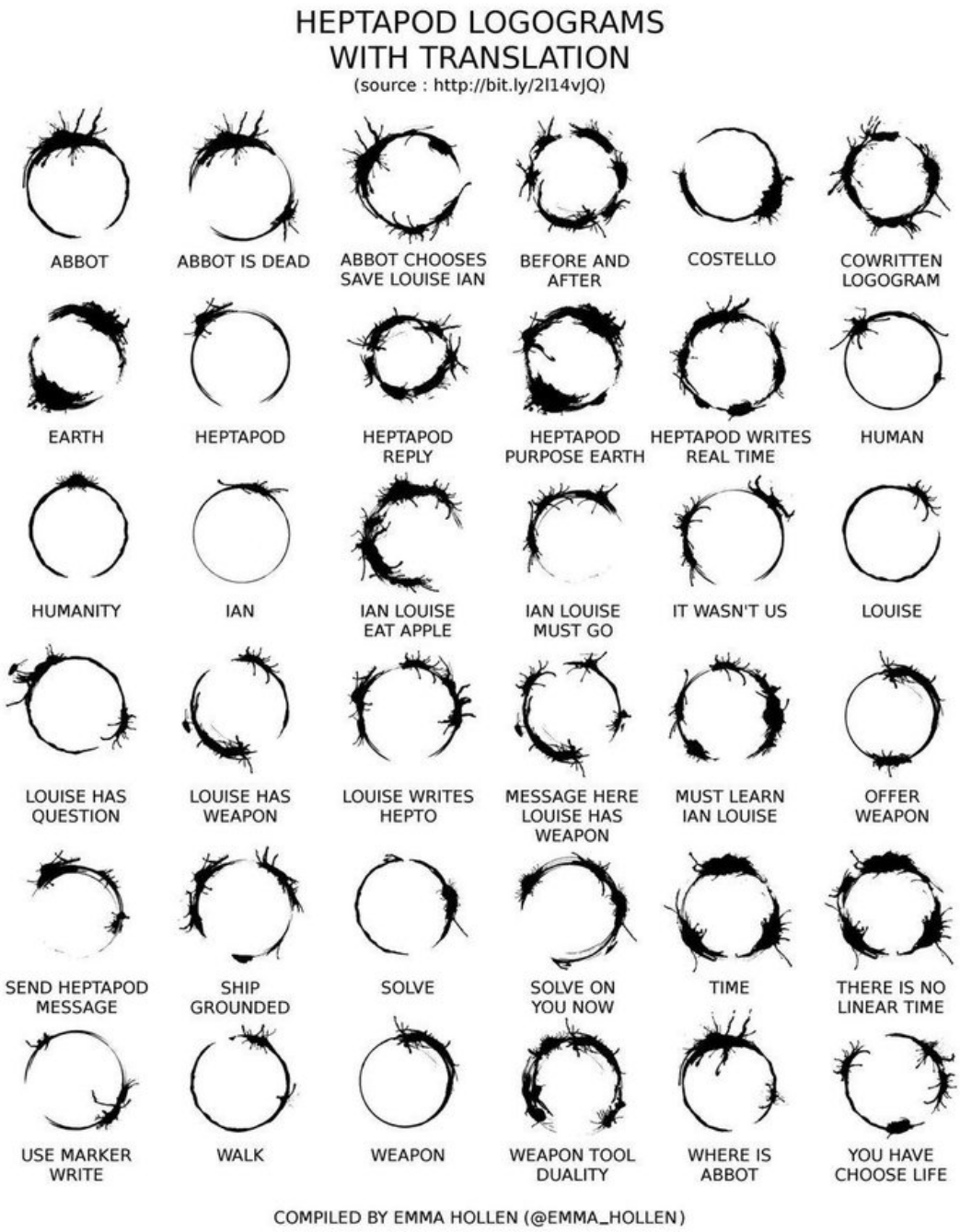
HashFormers: Towards Vocabulary-independent Pre-trained Transformers

The main contribution of this chapter is the peer-reviewed publication titled *HashFormers: Towards Vocabulary-independent Pre-trained Transformers* (Xue and Aletras, 2022), presented at the *Empirical Methods in Natural Language Processing* main conference in December 2022.

This work was largely inspired by movies (*Arrival, 2016*; *Persian Lessons, 2020*) and the words form an influential Chinese philosopher Zhuangzi (庄子) who lived around the 4th century BCE, which is also mentioned in Chapter 6, Vector Semantics and Embeddings in Jurafsky and Martin (2025).[1][2]

[1] 筌者所以在鱼，得鱼而忘筌。言者所以在意，得意而忘言。——《庄子》

[2] Nets are for fish; Once you get the fish, you can forget the net. Words are for meaning; Once you get the meaning, you can forget the words. *–Zhuangzi*



Logograms from the alien language in the movie *Arrival (2016)*. Image Source: https://x.com/1jcmaxwell/status/1166366420910968835

Abstract

Transformer-based pre-trained language models are vocabulary-dependent, mapping by default each token to its corresponding embedding. This one-to-one mapping results into embedding matrices that occupy a lot of memory (i.e. millions of parameters) and grow linearly with the size of the vocabulary. Previous work on on-device transformers dynamically generate token embeddings on-the-fly without embedding matrices using locality-sensitive hashing over morphological information. These embeddings are subsequently fed into transformer layers for text classification. However, these methods are not pre-trained. Inspired by this line of work, we propose HASHFORMERS, a new family of vocabulary-independent pre-trained transformers that support an unlimited vocabulary (i.e. all possible tokens in a corpus) given a substantially smaller fixed-sized embedding matrix. We achieve this by first introducing computationally cheap hashing functions that bucket together individual tokens to embeddings. We also propose three variants that do not require an embedding matrix at all, further reducing the memory requirements. We empirically demonstrate that HASHFORMERS are more memory efficient compared to standard pre-trained transformers while achieving comparable predictive performance when fine-tuned on multiple text classification tasks. For example, our most efficient HASHFORMER variant has a negligible performance degradation (0.4% on GLUE) using only 99.1K parameters for representing the embeddings compared to 12.3–38M parameters of state-of-the-art models.[3]

2.1 Introduction

The majority of transformer-based (Vaswani et al., 2017) pre-trained language models (PLMs; Devlin et al. 2019; Liu et al. 2019; Dai et al. 2019; Yang et al. 2019) are vocabulary-dependent, with each single token mapped to its corresponding vector in an embedding matrix. This one-to-one mapping makes it impractical to support out-of-vocabulary tokens such as misspellings or rare words (Pruthi et al., 2019; Sun et al., 2020a). Moreover, it linearly increases the memory requirements with the vocabulary size for the token embedding matrix (Chung et al., 2021). For example, given a token embedding size of 768, BERT-BASE with a vocabulary of 30.5K tokens needs 23.4M out of 110M total parameters, while ROBERTA-BASE with 50K tokens needs 38M out of 125M total parameters. Hence, disentangling the design of PLMs from the vocabulary size and tokenisation approaches would inherently improve memory efficiency and pre-training, especially for researchers with access to limited computing resources (Strubell et al., 2019a; Schwartz et al., 2020).

Previous efforts for making transformer-based models vocabulary-independent include dy-

[3]Code is available here: https://github.com/HUIYINXUE/hashformer and the pre-trained HashFormer models are available here: https://huggingface.co/klein9692.

namically generating token embeddings on-the-fly without embedding matrices using hash embeddings (Svenstrup et al., 2017; Ravi, 2019) over morphological information Sankar et al. (2021a). However, these embeddings are subsequently fed into transformer layers trained from scratch for on-device text classification without any pre-training. Without introducing any morphological information, Clark et al. (2022a) operated on Unicode characters and utilised a low-collision multi-hashing strategy to support ~1.1M Unicode codepoints as well as all possible permutations of four arbitrary Unicode codepoints, thus, can skip tokenisation explicitly while limiting the parameters of its embedding layer to 12.3M. Different to using hash embeddings, Xue et al. (2022) proposed models that take as input byte sequences representing characters without explicit tokenisation or a predefined vocabulary to pre-train transformers in multilingual settings.

In this paper, we propose HASHFORMERS a new family of vocabulary-independent PLMs. Our models support an unlimited vocabulary (i.e. all possible tokens in a given pre-training corpus) with a considerably smaller fixed-sized embedding matrix. We achieve this by employing simple yet computationally efficient hashing functions that bucket together individual tokens to embeddings inspired by the hash embedding methods of Svenstrup et al. (2017) and Sankar et al. (2021a). Our contributions are as follows:

1. To the best of our knowledge, this is the first attempt towards reducing the memory requirements of PLMs using various hash embeddings with different hash strategies, aiming to substantially reduce the embedding matrix compared to the vocabulary size;

2. Three HASHFORMER variants further reduce the memory footprint by entirely removing the need of an embedding matrix;

3. We empirically demonstrate that our HASHFORMERS are consistently more memory efficient compared to vocabulary-dependent PLMs while achieving comparable predictive performance when fine-tuned on a battery of standard text classification tasks.

2.2 Related Work

2.2.1 Tokenisation and Vocabulary-independent Transformers

Typically, PLMs are pre-trained on text that has been tokenised using subword tokenisation techniques such as WordPiece (Wu et al., 2016), Byte-Pair-Encoding (BPE; Sennrich et al. 2016) and SentencePiece (Kudo and Richardson, 2018).

Attempts to remove the dependency of PLMs on a separate tokenisation component include models that directly operate on sequences of characters (Tay et al., 2022b; El Boukkouri et al., 2020). However, these approaches do not remove the requirement of an embedding matrix.

Recently, Xue et al. (2022) proposed PLMs that take as input byte sequences representing characters without explicit tokenisation or a predefined vocabulary in multilingual settings. PLMs in Clark et al. (2022a) operating on Unicode characters or ngrams also achieved a similar goal. These methods improve memory efficiency but still rely on a complex process to encode the relatively long ngram sequences of extremely long byte/Unicode sequences, affecting their computational efficiency.

In a different direction, Sankar et al. (2021b) proposed PROFORMER, an on-device vocabulary-independent transformer-based model. It generates token hash embeddings (Svenstrup et al., 2017; Shi et al., 2009; Ganchev and Dredze, 2008) on-the-fly by applying locality-sensitive hashing over morphological features. Subsequently, hash embeddings are fed to transformer layers for text classification. However, PROFORMER is trained from scratch using task-specific data without any pre-training.

2.2.2 Compressing PLM Embeddings

A different line of work has focused on compressing the embedding matrix in transformer models (Ganesh et al., 2021). Prakash et al. (2020) proposed to use compositional code embeddings (Shu and Nakayama, 2018) to reduce the size of the embeddings in PLMs for semantic parsing. Zhao et al. (2021) developed a distillation method to align teacher and student token embeddings using a mixed-vocabulary training (i.e. the student and teacher models have different vocabularies) for learning smaller BERT models. However, these approaches still rely on a predefined vocabulary. Clark et al. (2022a) adopted a low-collision multi-hashing strategy to support ~1.1M Unicode codepoints and a larger space of character four-grams with a relatively small embedding matrix containing 12.3M parameters.

2.3 HashFormers

In this section, we present HASHFORMERS, a family of vocabulary-independent hashing-based pre-trained transformers.

2.3.1 Many-to-One Mapping from Tokens to an Embedding

Given a token t, HASHFORMERS use a hash function $\mathcal{H}$ to map t into a value v. Using hashing allows our model to map many tokens into a single embedding and support an infinite vocabulary. We obtain the embedding index by squashing its hash value v into $i = [1, ..., N]$ where $\mathbf{e} = \mathbf{E}_i$ is the corresponding embedding from a matrix $\mathbf{E} \in \mathbb{R}^{N \times d}$ where N is the number of the embeddings and d is their dimensionality. We assume that $|V| \gg N$ where $|V|$ is the size of the vocabulary. Subsequently, $\mathbf{e}$ is passed through a series of transformer layers for pre-training.

This is our first variant, HASHFORMER-Emb that relies on a look-up embedding matrix (see Figure 2.1). Our method is simple, vocabulary-free and independent of tokenisation choices.

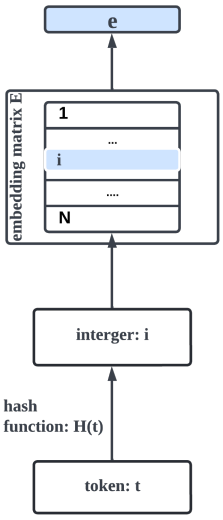


Figure 2.1: HASHFORMER-Emb.

2.3.2 Message-Digest Hashing (HashFormers-MD)

Our first approach to hash tokens is by using a Message-Digest (MD5) hash function (Rivest and Dusse, 1992) to map each token to its 128-bits output, $v = \mathcal{H}(t)$. The mapping can be reproduced given the same secret key. MD5 is a 'random' hashing approach, returning mostly different hashes for tokens with morphological or semantic similarities. For example:

$$\text{MD5}(\text{'play'}) = \textit{d077f244def8a70e5ea758bd8352fcd8}$$
$$\text{MD5}(\text{'plays'}) = \textit{4a258d930b7d3409982d727ddbb4ba88}$$

It is simple and does not require any pre-processing to obtain the bit encoding for each token. To map the hash output v into its corresponding embedding, we transform its binary value into decimal and then compute the index i to $\mathbf{E}$ as $i = v \,\%\, N$.

2.3.3 Locality-Sensitive Hashing (HASHFORMERS-LSH)

Locality-sensitive hashing (LSH) hashes similar tokens into the same indices with high probability (Rajaraman and Ullman, 2011). HASHFORMER-LSH uses LSH hashing to assign tokens with similar morphology (e.g. 'play', 'plays', 'played') to the same hash encoding. This requires an additional feature extraction step for token representation.

Token to Morphological Feature Vector: We want to represent each token with a vector $\mathbf{x}$ as a bag of morphological (i.e. character n-grams) features. For each token, we first extract

character n-grams ($n \in 1, 2, 3, 4$) to get a feature vector whose dimension is equal to the vocabulary size.[4] Each element in the feature vector is weighted by the frequency of the character n-grams of the token.

Morphological Vector to Hash Index: Once we obtain the morphological feature vector of each token, we first define N random hyperplanes, each represented by a random unit vector $\mathbf{r^i} \in \mathbb{R}^{d_x}$, where d_x is the dimensionality of the morphological feature vector. Following a similar approach to Kitaev et al. (2020), we compute the hash value v as the index of the nearest random hyperplane vector to the token's feature vector, $\mathbf{x}$ obtained by computing $v = \mathcal{H}(\mathbf{x}) = \arg\max(\mathbf{xR}), \mathbf{R} = [\mathbf{r}^1 ... \mathbf{r}^N]$ where $[\alpha\beta]$ denotes the concatenation of two vectors. This approach results into bucketing together tokens with similar morphological vectors. Similar to HASHFORMER-MD-Emb, we compute the embedding index as $i = v$.

To prevent storing a large projection matrix ($\mathbb{R}^{d_x \times N}$) for accommodating each unit vector, we design an on-the-fly computational approach leveraging the Circulant matrix. We only store a vector $\eta \in \mathbb{R}^{d_x}$ that is randomly initialized from the standard normal distribution, guaranteeing that each column $\mathbf{r}$ in the matrix $\mathbf{R}$ is a permutation of η with a unique offset value (e.g. $\mathbf{r}^1 = [\eta_2, ..., \eta_N, \eta_1]$). Each offset value only relies on the index of the hyperplane. This setting ensures that each hyperplane has the same L2-norm.

2.3.4 Compressing the Embedding Space

We also propose three embedding compression approaches that allow an even smaller number of parameters to represent token embeddings and support unlimited tokens (i.e. very large $|V|$) without forcing a large number of tokens to share the same embedding. For this purpose, we first use a hash function $\mathcal{H}$ to map each token t into a T-bit value b, $b \in [0, 2^T)$. Then, we pass b through a transformation procedure to generate the corresponding embedding (to facilitate computation, we cast b into a T-bit vector τ). We aim to ensure that tokens with different values b will be assigned to a different embedding by keeping the number of parameters relatively small. Figure 2.2 shows an overview of this method.

Pooling Approach (Pool) Inspired by Svenstrup et al. (2017) and Prakash et al. (2020), we first create a universal learnable codebook, which is a matrix denoted as $\mathbf{B} \in \mathbb{R}^{2^k \times d}$. Then, we split the hash bit vector τ in k successive bits without overlap to obtain $\lceil \frac{T}{k} \rceil$ binary values. We then cast these binary values into an integer value representing a codeword. Hence, each token is represented by a vector $\mathbf{c} \in \mathbb{R}^d$ with elements $c_j \in [0, 2^k)$. For example, given $k = 4$ and a 12-bits vector [1,0,1,0,0,1,0,0,0,0,0,1], 4-bit parts are treated as separate binary codewords [1010,0100,0001] then transformed into their decimal format codebook [10,4,1]. We construct

[4] We keep the top-50K most frequent n-grams in the pre-training corpus.

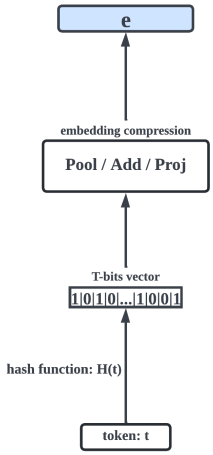


Figure 2.2: Compressing the embedding space.

the embedding $\mathbf{e} \in \mathbb{R}^d$ for each token by looking up the decimal codebook and extracting $\lceil \frac{T}{k} \rceil$ vectors corresponding to its $\lceil \frac{T}{k} \rceil$ codewords. We then apply a weighted average pooling on them using a softmax function:

$$\hat{\mathbf{W}}_j = \frac{\exp \mathbf{W}_j}{\sum_{l=1}^{\lceil \frac{T}{k} \rceil} \exp \mathbf{W}_l}, j = 1, .., \lceil \frac{T}{k} \rceil \tag{2.1a}$$

$$\mathbf{e} = \sum_{j=1}^{\lceil \frac{T}{k} \rceil} [\mathbf{B_c} \odot \hat{\mathbf{W}}]_j \tag{2.1b}$$

where $\mathbf{W} \in \mathbb{R}^{\lceil \frac{T}{k} \rceil \times d}$ is a learnable weight matrix as well as the codebook $\mathbf{B}$. The total number of parameters required for this pooling transformation is $(\lceil \frac{T}{k} \rceil + 2^k) \times d$. This can be much smaller than the $N \times d$ parameters required for standard PLMs that use a one-to-one mapping between tokens and embeddings, where $N = |V| \gg (\lceil \frac{T}{k} \rceil + 2^k)$. Figure 2.3 shows the overview of the Pool process.

Additive Approach (Add) Different to the Pool method that uses a universal codebook, we create T different codebooks $\{\mathbf{B}^1, \mathbf{B}^2, ..., \mathbf{B}^T\}$, each containing two learnable embedding vectors corresponding to codewords 0 and 1 respectively. We get a T-bits vector $\tau \in \{0, 1\}^T$ for each token, where each element in the vector τ is treated as a codeword. We look up each codeword in its corresponding codebook to obtain T vectors and add up them to compute the token embedding $\mathbf{e}$:

$$\mathbf{e} = \sum_{j=1}^{T} \mathbf{B}^j_{\tau_j} \Big/ \gamma \tag{2.2}$$

where $\mathbf{B}^j \in \mathbb{R}^{2 \times d}, j = 1, .., T$, γ is the scaling factor.[5] Hence, the total number of parameters the additive transformation approach requires is $2 \times T \times d$. Similar to the Pool approach, the

[5] Instead of averaging ($\gamma = T$), we set $\gamma = \sqrt{T}$ which we found to perform better in early experimentation.

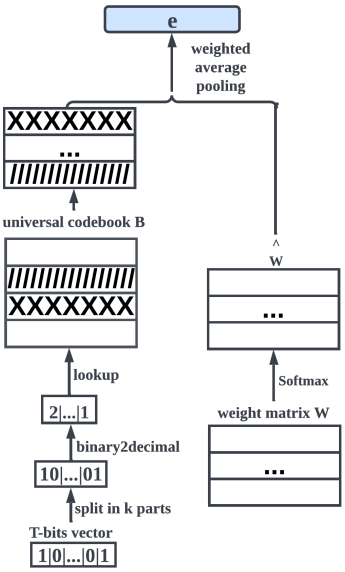


Figure 2.3: HASHFORMER-Pool.

number of parameters required is smaller than the vocabulary size: $2 \times T \times d \ll N = |V|$.

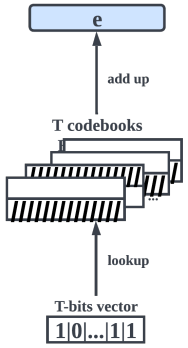


Figure 2.4: HASHFORMER-Add.

Projection Approach (Proj) Finally, we propose a new simpler approach compared to Pool and Add. We create T learnable random initialised vectors as T pseudo-axes to trace the orientation of each T-bits vector τ corresponding to the token t. Given a token bit vector τ, the j^{th} element in the embedding $\mathbf{e}$ is computed as the Pearson's correlation coefficient (PCC) between centered τ and the learnable vector $\mathbf{w}^j$ corresponding to j.

$$\begin{aligned} \mathbf{e}_j &= \frac{\langle \tau - \bar{\tau}, \mathbf{w^j} - \bar{\mathbf{w}}^{\mathbf{j}} \rangle}{\|\tau - \bar{\tau}\| \cdot \|\mathbf{w^j} - \bar{\mathbf{w}}^{\mathbf{j}}\|}, \quad j = 1, ..., d \\ \mathbf{e} &= (e_1, ..., e_d) \end{aligned} \tag{2.3}$$

$\bar{\tau}$ is the mean value of elements in τ, while $\bar{\mathbf{w}}^{\mathbf{j}}$ is the mean value of elements in $\mathbf{w}^{\mathbf{j}}$, for centering. $\mathbf{w}^j \in \mathbb{R}^d, j = 1, .., T$, hence, the total number of parameters the projection transfor-

mation approach requires is only $T \times d \ll N = |V|$. Figure 2.5 depicts an overview of our HASHFORMER-Proj model.

2.3.5 Hashing for Compressed Embeddings

Similar to the embedding-based HASHFORMERS-Emb, our embedding compression-based models also consider the same two hash approaches (MD and LSH) for generating the T-bit vector of each token.

MD5: We directly map the tokens to its 128-bits output b with a universal secret key.

LSH: We repeat the same morphological feature extraction step to obtain a feature vector $\mathbf{x}$ corresponding to each token t. However, rather than using 2^T random hyperplanes that require storing vectors of size $\mathbb{R}^{2^T}$, we simply use T random hyperplanes similar to Ravi (2019); Sankar et al. (2021b). Each bit in b represents which side of the corresponding hyperplane $\mathbf{r} \in \mathbb{R}^d$ the feature vector $\mathbf{x}$ is located: $b_j = \text{sgn}(\text{sgn}(\mathbf{x} \cdot \mathbf{r^i}) + 1), j = 1, ..., T$. This allows an on-the-fly computation without storing any vector (Ravi, 2019).

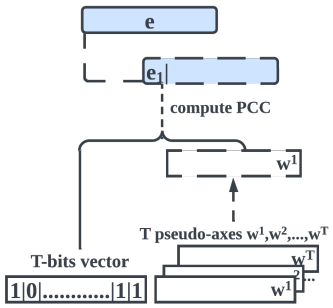


Figure 2.5: HASHFORMER-Proj.

2.3.6 Pre-training Objective

Since our models support an arbitrary number of unique tokens, it is intractable to use a standard Masked Language Modeling (Devlin et al., 2019) pre-training objective. We opted using SHUFFLE + RANDOM (S+R), a computationally efficient three-way classification objective introduced by Yamaguchi et al. (2021) for predicting whether tokens in the input have been shuffled, replaced with random tokens or remain intact.

2.4 Experimental Setup

2.4.1 Baseline Models

We compare HASHFORMERS against the following baselines: (i) a BERT-base model (Devlin et al., 2019) with BPE tokenisation and an MLM objective (BERT-MLM); (ii) another BERT-base model with BPE tokenisation and a Shuffle+Random objective (BERT-S+R); (iii) CANINE-C[6] (Clark et al., 2022a) a vocabulary-free pre-trained PLM on Unicode character sequences; (iv) PROFORMER[7] (Sankar et al., 2021b) a vocabulary-free LSH projection based transformer model with two encoder layers that is not pre-trained but only trained from scratch on the task at hand.

2.4.2 Implementation Details

Model Architecture Following the architecture of BERT-base, we use 12 transformer layers, an embedding size of 768 and a maximum sequence length of 512.[8] For HASHFORMERS-LSH, we set $T = 128$ to make it comparable to HASHFORMERS-MD, as MD5 produces a 128-bit hash value. For HASHFORMER-MD-Pool and HASHFORMER-LSH-Pool, we choose $k = 10$ to keep the number of total parameters for the embeddings relatively small. We also experiment with two sizes of the embedding matrix of HASHFORMERS-Emb for MD and LSH hashing. The first uses an embedding matrix of 50K, the same number of embedding parameters as BERT-base, while the second uses 1K which is closer to the size of the smaller Pool, Add and Proj models.

Hyperparameters Hyperparameter selection details are in section 2.7.

Pre-training We pre-train all HASHFORMERS, BERT-MLM and BERT-S+R on the English Wikipedia and BookCorpus (Zhu et al., 2015) from HuggingFace (Lhoest et al., 2021) for up to 500k steps with a batch size of 128. For our HASHFORMER models, we use white space tokenisation resulting into a vocabulary of 11,890,081 unique tokens. For BERT-MLM and BERT-S+R, we use a 50,000 BPE vocabulary Liu et al. (2019).

Hardware For pre-training, we use eight NVIDIA Tesla V100 GPUs. For fine-tuning on downstream tasks, we use one NVIDIA Tesla V100 GPU.

[6] We use the off-the-shelf CANINE-C from https://huggingface.co/google/canine-c.

[7] ProFormer is not open-source, hence we have re-implemented it following the description of the model in the paper.

[8] We note that the transformer encoder could easily be replaced with any other encoder.

2.4.3 Predictive Performance Evaluation

We evaluate all models on GLUE (Wang et al., 2018) benchmark. We report matched accuracy for MNLI, Matthews correlation for CoLA, Spearman correlation for STS, F1 score for QQP and accuracy for all other tasks.

Model	Token	MNLI↑	QNLI↑	QQP↑	RTE↑	SST↑	MRPC↑	CoLA↑	STS↑	Avg.↑
BERT-MLM	subword	**81.9**	**88.9**	86.7	60.4	**92.0**	85.7	54.5	86.0	79.5(0.4)
BERT-S+R	subword	79.9	88.7	86.7	64.6	88.6	85.6	55.6	**86.8**	79.6(0.3)
CANINE-C	Unicode	77.7	87.6	82.8	62.0	85.7	81.4	2.3	83.9	70.4(1.3)
ProFormer	word	45.2	59.1	71.4	53.9	82.1	71.2	9.7	22.1	51.8(0.5)
HashFormers-MD (Ours)										
Emb (50K)	word	79.6	88.4	**86.9**	**66.4**	88.0	**86.8**	57.3	86.1	**79.9(0.3)**
Emb (1K)	word	67.9	80.5	81.0	55.8	72.9	78.4	19.0	79.0	66.8(0.9)
Pool	word	75.6	84.9	84.9	59.7	86.7	82.7	45.7	82.0	75.3(0.2)
Add	word	76.2	86.3	85.2	60.2	86.6	81.9	47.4	82.2	75.7(0.5)
Proj	word	76.0	85.8	84.8	60.9	87.3	83.0	45.9	82.1	75.7(0.3)
HashFormers-LSH (Ours)										
Emb (50K)	word	76.1	86.5	85.5	65.5	83.6	84.2	42.7	83.7	76.0(0.3)
Emb (1K)	word	65.6	80.1	80.0	56.4	71.3	78.1	5.2	76.9	64.2(0.8)
Pool	word	78.0	87.7	86.4	65.6	88.1	84.2	55.3	85.6	78.9(0.3)
Add	word	78.6	88.2	86.0	63.1	88.0	84.0	**57.7**	85.9	78.9(0.2)
Proj	word	79.2	88.7	86.5	63.4	88.9	84.6	56.2	85.5	79.1(0.3)

Table 2.1: Results on GLUE dev sets with standard deviations over three runs in parentheses. **Bold** values denote best performing method in each task. ↑ and ↓ denote that higher and lower values are preferred, respectively.

2.4.4 Efficiency Evaluation

Furthermore, we use the following metrics to measure and compare the memory and computational efficiency of HASHFORMERS and the baselines.

Memory Efficiency Metrics We define the three memory efficiency metrics together with a performance retention metric to use it as a point of reference:

- **Performance Retention Ratio:** We compute the ratio between the predictive performance of our target model compared to a baseline model performance. A higher PRR indicates better performance.

$$\mathbf{PRR} = \frac{\mathbf{score}_{\text{Model}}}{\mathbf{score}_{\text{Baseline}}} \tag{2.4}$$

- **Parameters Compression Ratio (All):** We compute use the ratio between the total

number of parameters of our target model and that of the baseline to measure the memory efficiency of the target model compared to the baseline. A higher PCR_{All} score indicates better memory efficiency for the entire model.

$$\mathbf{PCR_{All}} = 1 - \frac{\#\text{Model Params}}{\#\text{Baseline Params}} \tag{2.5}$$

- **Parameters Compression Ratio (Emb):** We also use the ratio between the number of parameters required by a target model for representing embeddings and that of the baseline. A higher PCR_{Emb} score indicates better memory efficiency for the embedding representation.

$$\mathbf{PCR_{Emb}} = 1 - \frac{\#\text{Model Emb Params}}{\#\text{Baseline Emb Params}} \tag{2.6}$$

- **Proportion of Embedding Parameters:** We also use the proportion of parameters of embeddings out of the total parameters of each model to show the memory footprint of the embedding space on each model.

$$\mathbf{PoEP} = \frac{\#\text{Emb Params}}{\#\text{Total Params}} \tag{2.7}$$

Ideally, we expect a smaller PoEP, indicating that the embedding parameters occupy as little memory as possible out of the total number of parameters of a model.

For the number of parameters calculations, please see section 2.8.

Computational Efficiency Metrics We also measure the computational efficiency for pre-training (PT) and inference (Infer). Each pre-training step is defined as a forward pass and a backward pass. The inference is defined by a single forward pass.

- **Time per Sample (Time)** This measures the average time of a sample completing a PT or Infer step. It is measured in milliseconds (ms)/sample. Lower PT and Infer time indicate a more computationally efficient model.

- **Speed-up Rate** We finally measure the model's computation speed-up rate against a baseline. It is defined as:

$$\mathbf{Speed\text{-}upRate} = 1 \Big/ \frac{\mathbf{Time}_{\text{Model}}}{\mathbf{Time}_{\text{Baseline}}} \tag{2.8}$$

2.5 Results

2.5.1 Predictive Performance Comparison

Table 2.1 presents results on GLUE for our HASHFORMERS models and all baselines. We first observe that both the performance of our HASHFORMERS-Emb models (MD and LSH) are

comparable to the two BERT variants (MLM and S+R) and CANINE-C on average GLUE score (79.9 and 76.0 vs. 79.5, 79.6 and 70.4 respectively). Surprisingly, the more sophisticated HASHFORMER-LSH-Emb that takes morphological similarity of tokens into account does not outperform HASHFORMER-MD-Emb that uses a random hashing. We believe that HASHFORMER-MD generally outperforms HASHFORMER-LSH mainly due to its ability to map morphologically similar tokens to different vectors. This way it can distinguish tenses etc. On the other hand, HASHFORMER-LSH confuses words with high morphological similarity (e.g. play, played) because it assigns them to the same embedding.

However, LSH contributes to the performance improvement of smaller HASHFORMERS with compressed embedding spaces compared to their MD variants, i.e. Add (78.9 vs. 75.3), Add (78.9 vs. 75.7) and Proj (79.1 vs. 75.7). The best performing compressed HASHFORMER-LSH-Proj model obtains 79.1 average GLUE score, which is only 0.4 lower than the BERT baselines. Reducing the number of embedding vectors in Emb (1K) models is detrimental to performance and leads to drastic drops between 11.8% and 13.1%. This indicates that the model size plays a more important role than the choice of tokenisation approach (i.e. white space or BPE) or the vocabulary size (i.e. 12M vs. 50K). At the same time, compared to Emb, the Pool, Add and Proj approaches do not suffer from predictive accuracy degradation, i.e. 0.4-4.2%.

All our HASHFORMERS show clear advantages compared to the LSH based PROFORMER, which is not pre-trained across the majority of tasks (i.e. MNLI, QNLI, QQP, MRPC, CoLA and STS). Although PROFORMER shows that for a relatively simpler sentiment analysis task (SST), pre-training might not be necessary.

2.5.2 Memory Efficiency Comparison

Table 2.2 shows the results on memory efficiency and performance retention (%) on GLUE using BERT-MLM as a baseline. Notably, Pool, Add and Proj models provide large compression to the total number of embeddings parameters compared to Emb as well as CANINE-C and BERT variants. This is approximately a 30% PCR_{All} and 97-99% PCR_{Emb} compared to BERT. These models also achieve very high performance retention (from 94.7% to 99.5%) which highlights their efficiency. In one case, HASHFORMER-LSH-Add outperforms the BERT-MLM baseline on CoLA with a retention ratio of 105.9% using only 197.4K parameters for token embeddings.

Proj variants, the smallest of HASHFORMERS achieve the highest performance retention (95.2% with MD, 99.5% with LSH) compared to Pool (94.7% with MD, 99.2% with LSH) and Add (95.2% with MD, 99.2% with LSH). Overall, they only have a negligible drop in performance retention (0.5%) while they are extremely more memory efficient. Proj uses a substantially smaller number of embedding parameters (99.1K) compared to CANINE-C and BERT variants (i.e., 12.3M and 38.6M respectively). In general, Pool, Add and Proj models lead to a 30%

Emb.	MNLI↑	QNLI↑	QQP↑	RTE↑	SST↑	MRPC↑	CoLA↑	STS↑	GLUE Avg.↑	#Total Params↓	#Emb Params↓	PCR↑ (All)	PCR↑ (Emb)	PoEP↓
CANINE-C	94.9	98.5	95.5	102.6	93.2	95.0	4.2	97.6	88.6	121.0M	12.3M	2.9	68.1	10.2
ProFormer	55.2	66.5	82.4	89.2	89.2	83.1	17.8	25.7	65.2	**15.1M**	322.6K	**87.9**	99.2	2.1
HashFormers-MD (Ours)														
Emb (50K)	**97.2**	99.4	**100.2**	**109.9**	95.7	**101.3**	105.1	**100.1**	**100.5**	124.6M	38.6M	0.0	0.0	31.0
Emb (1K)	82.9	90.6	93.4	92.4	79.2	91.5	34.9	91.9	84.0	86.8M	797.2K	30.3	97.9	1.0
Pool	92.3	95.5	97.9	98.8	94.2	96.5	83.9	95.3	94.7	86.8M	797.2K	30.3	97.9	1.0
Add	93.0	97.1	98.3	99.7	94.1	95.6	87.0	95.6	95.2	86.2M	197.4K	30.8	99.5	0.2
Proj	92.8	96.5	97.8	100.8	94.9	96.8	84.2	95.5	95.2	86.1M	**99.1K**	30.9	**99.7**	**0.1**
HashFormers-LSH (Ours)														
Emb (50K)	92.9	97.3	98.6	108.4	90.9	98.2	78.3	97.3	95.6	124.6M	38.6M	0.0	0.0	31.0
Emb (1K)	80.1	90.1	92.3	93.4	77.5	91.1	9.5	89.4	80.8	86.8M	797.2K	30.3	97.9	1.0
Pool	95.2	98.7	99.7	108.6	95.8	98.2	101.5	99.5	99.2	86.8M	797.2K	30.3	97.9	1.0
Add	96.0	99.2	99.2	104.5	95.7	98.0	**105.9**	99.9	99.2	86.2M	197.4K	30.8	99.5	0.2
Proj	96.7	**99.8**	99.8	105.0	**96.6**	98.7	103.1	99.4	99.5	86.1M	**99.1K**	30.9	**99.7**	**0.1**

Table 2.2: Memory efficiency metrics and performance retention (%) on GLUE for HASHFORMER models, CANINE-C and ProFormer using BERT-MLM as a baseline. ↑ and ↓ denote that higher and lower values are preferred, respectively.

Model	PT Time (ms/samp)↓	PT Speed-up Rate↑	Infer Time (ms/samp)↓	Infer Speed-up Rate↑
BERT				
-MLM	24.9	1.0	4.6	1.0
-S+R	11.6	2.1x	4.6	1.0x
CANINE-C	-	-	6.9	0.6x
HASHFORMERS (Ours)				
-Emb	11.6	2.1x	2.0~4.6	1.0x~2.4x
-Pool	12.0	2.1x	2.0~4.6	1.0x~2.3x
-Add	11.7	2.1x	2.0~4.6	1.0x~2.4x
-Proj	10.6	2.4x	1.8~4.6	1.0x~2.6x

Table 2.3: Results on pre-training speed and inference speed under different embedding compression strategies. We use BERT-MLM as the baseline model. The sequence length is fixed to 512 for pre-training. For inference, sequence length is equal to the length of the longest sequence in the batch. ↑ and ↓ denote that higher and lower values are preferred, respectively.

reduction in the total number of parameters (around 30.0M) compared to the baseline model and make their embedding footprint minimal, i.e. 0.1-1% PoEP. On the other hand, CANINE-C has a larger embedding footprint (10.2% PoEP) but with similar or smaller performance retention compared to HASHFORMERS.

HASHFORMERS-Emb have an embedding matrix of equal size (i.e. 50K embeddings) as BERT. However, BERT only supports a vocabulary of 50K tokens, while HASHFORMERS-Emb

supports an unlimited vocabulary, e.g. 12M unique tokens in our pre-training corpora. Using a smaller embedding matrix (i.e. 1K), the performance retention drops 20%~26%. Despite the fact that HASHFORMERS-Emb (1K) has a similar number of embedding parameters as the embedding compression approaches (i.e. Pool, Add, Proj), it falls far behind those models, i.e. between 8.5% and 14.3% for both MD and LSH variants. This demonstrates the effectiveness of our proposed embedding compression approaches.

Although the more lightweight ProFormer with only two transformer layers consists of 15.1M parameters in total (approximately 87.9% PCR_{All}), its performance[9] fall far behind our worst HASHFORMER-MD-Pool with a difference of 29.5% PRR on GLUE Avg. score. Nevertheless, ProFormer requires more bits for hashing the tokens, resulting in more parameters for representing token embeddings (322.6K) compared to HASHFORMERS-Add and HASHFORMERS-Proj (197.4K and 99.1K). Such memory efficiency gains substantially sacrifice model's predictive performance.

2.5.3 Computational Efficiency Comparison

Table 2.3 shows the pre-training (PT) and inference (Infer) time per sample for HASHFORMERS, CANINE-C, BERT-S+R using BERT-MLM as a baseline for reference. We note that HASHFORMERS have comparable pre-training time (PT) to the fastest BERT model (BERT-S+R). This highlights that the complexity of the pre-training objective is more important than the size of the embedding matrix for improving computational efficiency for pre-training.

During inference, we observe that the speed-up obtained by HASHFORMERS is up to 2.6x compared to both BERT models. However, this is due to the tokenisation approach. HASHFORMERS operate on the word level, so the sequence length of the input data is smaller, leading to inference speed-ups. Finally, we observe that CANINE-C has a slower inference time compared to both BERT models and HASHFORMERS. This might be due to its relatively more complex approach for processing the long Unicode character input sequence.

2.6 Conclusions

We have proposed HASHFORMERS, a family of vocabulary-independent hashing-based pre-trained transformers. We have empirically demonstrated that our models are computationally cheaper and more memory efficient compared to standard pre-trained transformers, requiring only a fraction of their parameters to represent token embeddings. HASHFORMER-LSH-Proj variant needs 99.1K parameters for representing the embeddings compared to millions of parameters required by state-of-the-art models with only a negligible performance degradation.

[9]The predictive performance of ProFormer does not improve, even if we train it for four times more epochs (20 epochs). We report the results when trained for a maximum of five epochs.

For future work, we plan to explore multilingual pre-training with HASHFORMERS and explore their ability in encoding linguistic properties Alajrami and Aletras (2022).

Limitations

We experiment only using English data to make comparisons with previous work easier. For languages without explicit white spaces (e.g. Chinese and Japanese), our methods can be applied with different tokenisation techniques, e.g. using a fixed-length window of characters.

Acknowledgments

This project made use of time on Tier 2 HPC facility JADE2, funded by EPSRC (EP/T022205/1). We would like to thank Miles Williams and the anonymous reviewers for their invaluable feedback.

2.7 Appendix A: Hyperparameters

The hyperparameters used in pre-training are listed in Table 2.4.

Hyperparameter	Pretraining
Maximum train epochs	10 epochs
Batch size (per GPU)	16 instances
Adam ϵ	1e-8
Adam β_1	0.9
Adam β_2	0.9999
Sequence length	512
Peak learning rate	1e-4 for MLM, 5e-5 for others
Learning rate schedule	linear
Warmup steps	10000
Weight decay	0.01
Attention Dropout	0.1
Dropout	0.1

Table 2.4: Details of hyperparameters used in pre-training.

The hyperparameters used in fine-tuning are listed in Table 2.5.

Hyperparameter	Fine-tuning
Maximum train epochs	5 epochs
Batch size (per GPU)	32 instances
Adam ϵ	1e-6
Adam β_1	0.9
Adam β_2	0.999
Peak learning rate	3e-5
Learning rate schedule	cosine with hard restarts
Warmup steps	first 6% steps
Weight decay	0
Attention Dropout	0.1
Dropout	0.1
Evaluation steps	2455 for MNLI, 655 for QNLI, 2275 for QQP, 48 for RTE, 421 for SST, 69 for MRPC, 162 for CoLA and 108 for STS

Table 2.5: Details of hyperparameters used in fine-tuning.

2.8 Appendix B: Model Parameter Counts

We count the total number of parameters of each model on a binary classification task. This is computed by counting all learnable variables used for the task (including those in the classification head) without freezing any weights. For all BERT variants and our HASHFORMERS, we

adopt the same setting of BERT-Base by setting $\text{Dim}_{\text{hidden}} = 768$, $\text{Dim}_{\text{intermediate}} = 3072$ with 12 hidden layers and 12 attention heads. For CANINE-C, we use the default base-sized model whose $\text{Dim}_{\text{hidden}} = 768$, $\text{Dim}_{\text{intermediate}} = 3072$ and has 12 hidden layers and attention heads.

We only count the number of parameters which are used for retrieving or generating the embeddings of any tokens (excluding those special tokens e.g. <PAD>) and we also exclude those for position embeddings. Specifically, #[Model] Emb Params are computed as the follow:

- **BERT variants**:

$$\#\text{BERT} = |V| \times d \tag{2.9}$$

- **CANINE-C**:

$$\#\text{CANINE-C} = \#\text{Hash Buckets} \times d \tag{2.10}$$

  CANINE-C employs 16,000 hash buckets (Clark et al., 2022a).

- **PROFORMER**:

$$\#\text{ProFormer} = \#\text{LSH Digest Size} \times d \tag{2.11}$$

  PROFORMER hashes each token into a 420-bit vector (Sankar et al., 2021b).

- **HASHFORMERS-Emb**:

$$\#\text{HashFormers-Emb} = N \times d \tag{2.12}$$

- **HASHFORMERS-Pool**:

$$\#\text{HashFormers-Pool} = (\lceil \frac{T}{k} + 2^k \rceil) \times d \tag{2.13}$$

- **HASHFORMERS-Add**:

$$\#\text{HashFormers-Add} = 2 \times T \times d \tag{2.14}$$

- **HASHFORMERS-Proj**:

$$\#\text{HashFormers-Proj} = T \times d \tag{2.15}$$

2.9 Appendix C: Activation Memory Analysis

By retaining vanilla BERT embeddings, HASHFORMERS-Emb avoid any increase in activation memory consumption. Other HASHFORMERS variants achieve similar efficiencies by bypassing the storage of additional activation memory for embedding layers. Specifically, HASHFORMERS-Add leverages the EmbeddingBag technique—computing the reduction of a "bag" of embeddings directly without storing intermediate results (Naumov et al., 2019; Paszke et al., 2019). This

efficiency can be extended to HASHFORMERS-Pool by optimising the kernel to support channel-wise weighted summation reduction. Similarly, HASHFORMERS-Proj can minimise activation memory through the use of multi-channel convolution or customised fused kernels.

While storing token indices typically incurs negligible memory costs (Korthikanti et al., 2023), HASHFORMERS-Add and HASHFORMERS-Pool increase this footprint by a factor of 13, while HASHFORMERS-proj increases it 128-fold. However, this is mitigated in practice as these variants do not require long-integer storage. Further optimisation depends on modern hardware support (e.g., GPUs), a requirement shared by extreme low-bit quantisation methods in language model training and inference (Ma et al., 2025, 2024). Additionally, we observe that activation memory in the encoder layers could be reduced by 0% to 50% by leveraging English whitespace tokenisation, which can merge 1–3 subwords to shorten the overall input sequence length.

2.10 Appendix D: Potential on Multilingualism and Non-English Languages

The encoder-only multilingual model XLM-R (Conneau et al., 2020) uses a shared vocabulary across languages; however, its limited parameter budget for the embedding layer often results in sampling bias during vocabulary construction. XLM-V (Liang et al., 2023) addresses this "vocabulary bottleneck" and the resulting out-of-vocabulary (OOV) issues by significantly scaling the vocabulary size from 250K to 1M tokens.

We hypothesise that HASHFORMERS-LSH can strike an effective balance between vocabulary size and predictive performance. Because of distinct morphological differences between languages, tokens are unlikely to be assigned similar hash embeddings, thereby reducing collisions. Furthermore, our analysis in subsection 2.4.4 and section 2.9 suggests that HASHFORMERS, when paired with full-word tokenisation, can enhance inference efficiency. By mitigating the over-segmentation common in multilingual models with constrained vocabularies, this approach effectively reduces sequence lengths for non-English inputs.

For languages that do not use whitespace boundaries, such as Chinese and Japanese, HashFormers can be integrated with language-specific tokenisers like Jieba[10] or MeCab[11].

2.11 Appendix E: HASHFORMERS with BPE Tokenisation

Table 2.6 presents results on GLUE for our HASHFOMERS with BPE tokenisation. In general, we observe that using BPE tokenisation, the performance of HASHFOMERS slightly drops.

[10] https://github.com/fxsjy/jieba
[11] https://taku910.github.io/mecab/

Model	Token	MNLI↑	QNLI↑	QQP↑	RTE↑	SST↑	MRPC↑	CoLA↑	STS↑	Avg.↑
HashFormers-MD										
Emb (50K)	subword	78.6	87.7	86.0	65.6	88.5	85.1	51.2	85.0	78.5(0.4)
Emb (50K)	word	79.6	88.4	86.9	66.4	88	86.8	57.3	86.1	79.9(0.3)
Proj	subword	74.6	84.8	83.7	58.7	85.5	80.7	44.6	80.1	74.1(0.5)
Proj	word	76.0	85.8	84.8	60.9	87.3	83.0	45.9	82.1	75.7(0.3)
HashFormers-LSH										
Emb (50K)	subword	62.6	80.2	80.8	59.3	71.3	80.2	18.3	75.5	66.0(0.2)
Emb (50K)	word	76.1	86.5	85.5	65.5	83.6	84.2	42.7	83.7	76.0(0.3)
Proj	subword	78.2	87.5	86.3	64.3	88.6	85.5	51.2	85.1	78.3(0.1)
Proj	word	79.2	88.7	86.5	63.4	88.9	84.6	56.2	85.5	79.1(0.3)

Table 2.6: Results on GLUE dev sets with standard deviations over three runs in parentheses using BPE tokenisation. ↑ and ↓ denote that higher and lower values are preferred, respectively.

Chapter 3

Publication II

Pit One Against Many: Leveraging Attention-head Embeddings for Parameter-efficient Multi-head Attention

The main contribution of this chapter is the peer-reviewed publication titled *Pit One Against Many: Leveraging Attention-head Embeddings for Parameter-efficient Multi-head Attention* (Xue and Aletras, 2023), presented at the *Findings of the Association for Computational Linguistics: EMNLP 2023* in December 2023.

This work draws inspiration from the film *Ant-Man and the Wasp: Quantumania (2023)*, specifically its creative portrayal of the quantum principle of superposition as a "probability storm". In this storm, every potential choice Ant-Man could make is visualised as a different version of himself, creating an entire army. The principle of superposition posits that an object can exist in multiple states at the same time. It's only when a specific choice is "measured", or actualised, that these possibilities collapse into a single, definitive state. "What if we treated token embeddings at different positions and the hidden states from various attention heads as if they were quantum superpositions?", Huiyin was soul searching...[1][2]

[1] 他山之石，可以攻玉。——《诗经》

[2] A stone taken from another mountain may serve as a tool to polish the local jade. –*The Book of Songs*

A poster of the movie: *Ant-Man and the Wasp: Quantumania (2023)*. Image Source: https://www.egames.news/entretenimiento/Marvel-Explicacion-del-final-de-Ant-Man-and-the-Wasp-Quantumania-20230218-0012.html

Abstract

Scaling pre-trained language models has resulted in large performance gains in various natural language processing tasks but comes with a large cost in memory requirements. Inspired by the position embeddings in transformers, we aim to simplify and reduce the memory footprint of the multi-head attention (MHA) mechanism. We propose an alternative module that uses only a single shared projection matrix and multiple head embeddings (MHE), i.e. one per head. We empirically demonstrate that our MHE attention is substantially more memory efficient compared to alternative attention mechanisms while achieving high predictive performance retention ratio to vanilla MHA on several downstream tasks. MHE attention only requires a negligible fraction of additional parameters ($3nd$, where n is the number of attention heads and d the size of the head embeddings) compared to a single-head attention, while MHA requires $(3n^2 - 3n)d^2 - 3nd$ additional parameters.[3]

[3]Code: https://github.com/HUIYINXUE/simpleMHE

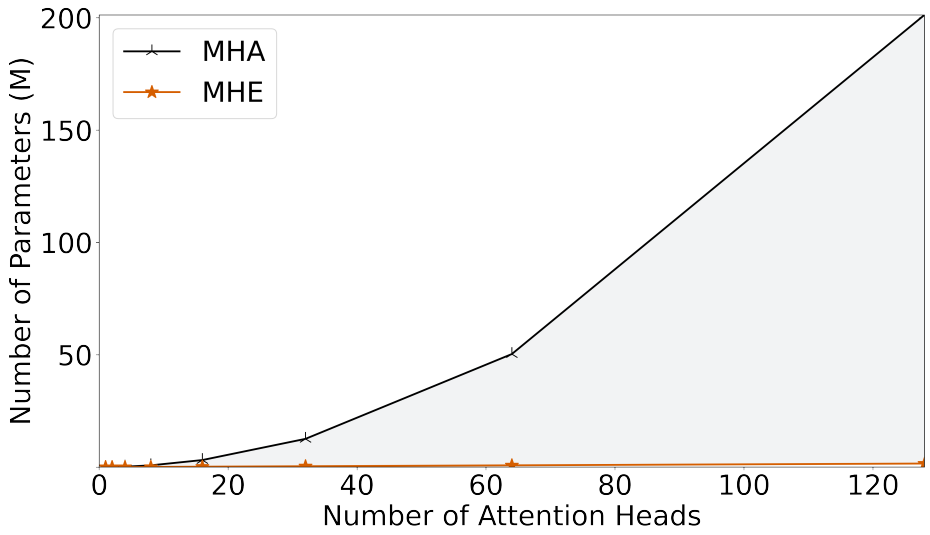


Figure 3.1: Number of parameters for an attention sublayer and different number of attention heads using multi-head attention MHA and our multi-head embedding attention MHE. We fix the dimension of attention to 64, only counting the parameters for projecting queries, keys, and values.

3.1 Introduction

Scaling pre-trained language models (PLMs) aims to enhance performance by increasing their size and capacity, leading to models with an unprecedented number of parameters (Kaplan et al., 2020; Chowdhery et al., 2022; Hoffmann et al., 2022). Just by increasing the size of PLMs and the pre-training data has yielded state-of-the-art performance on various natural language processing (NLP) tasks (Devlin et al., 2019; Liu et al., 2019; Clark et al., 2020; Raffel et al., 2020; Brown et al., 2020; Clark et al., 2022a; Ouyang et al., 2022; Touvron et al., 2023).

However, the pursuit of developing larger PLMs comes with large computational requirements. This has direct environmental implications such as large carbon emissions (Lacoste et al., 2019; Strubell et al., 2019b; Weidinger et al., 2022), conflicting with the principles of Green artificial intelligence development (Schwartz et al., 2020). Moreover, scaling can hinder researchers with limited access to computing resources to participate in advancing the field (Schwartz et al., 2020). This results in inequalities, where only a privileged few can actively contribute, potentially impeding diversity and inclusivity (Weidinger et al., 2022).

The backbone of transformers (Vaswani et al., 2017) is the multi-head attention (MHA) module that extends the standard single-head attention (SHA) proposed by Cho et al. (2014). MHA applies an attention mechanism (i.e. head) multiple times for the same set of queries, keys and values by using a different set of parameters (i.e. projection matrices) for each of them.

This results in MHA modules with a large memory footprint that increases with the number of layers and attention heads per layer in PLMs (Devlin et al., 2019; Brown et al., 2020; Ouyang et al., 2022; Touvron et al., 2023). Figure 3.1 shows how the number of parameters of a single attention sublayer increases with its number of attention heads.

Previous work has attempted to address this issue by proposing to share projection matrices or eliminating them entirely to improve the parameter efficiency of MHA. Lan et al. (2020a) proposed sharing projection parameters for keys, queries and values across layers, while Kitaev et al. (2020) introduced a method for sharing the projection matrix between keys and values within each transformer layer. Additionally, similar approaches use a multi-query attention approach that uses a pair of global projection matrices for keys and values in each layer (Shazeer, 2019; Chowdhery et al., 2022; Ainslie et al., 2023b). Furthermore, Yan et al. (2021) eliminate the projection matrices entirely and directly treat the input hidden states as both keys and values. In a different direction, Lee-Thorp et al. (2022) propose models that replace the attention blocks with token-mixture blocks (i.e. using linear or Fourier transformations) that contain fewer or no parameters compared to MHA.

Inspired by the position embeddings in transformers (Vaswani et al., 2017; Devlin et al., 2019), we aim to simplify and reduce the memory footprint of the MHA mechanism. We achieve this using only a single projection matrix for each of the keys, queries and values respectively shared across all attention heads, and one embedding per head (MHE).

Our contributions are as follows:

- We propose MHE, a novel attention module that uses shared projection matrices across heads that are modified by corresponding embedding heads. Our method generates multiple attention heads requiring only a small fraction of additional parameters compared to single-head attention.
- We empirically demonstrate that our MHE attention is substantially more parameter efficient compared to alternative attention mechanisms while achieving high predictive performance retention ratio (i.e. 92.9~98.7%) to MHA on several downstream tasks. MHE is $(3n^2 - 3n)d^2 - 3nd$ smaller than MHA for a single attention sublayer with n attention heads and a hidden dimension of d per head.

3.2 Related Work

3.2.1 Model Compression

To make PLMs memory efficient, previous work has focused on the following post-hoc model compression approaches (Ganesh et al., 2021; Tay et al., 2022a).

Quantisation Hubara et al. (2017) proposed representing weights using fewer bits to reduce memory requirements. Zadeh et al. (2020) introduced a method for identifying the outliers in weights and excluded them during quantisation. Another direction involves additional training steps to adjust the quantised weights, i.e., quantisation-aware training (Zafrir et al., 2019; Boo and Sung, 2020; Stock et al., 2020; Shen et al., 2020; Tambe et al., 2021; Tao et al., 2022). Bai et al. (2022) developed a more efficient post-training quantisation approach that minimises the reconstruction error incurred by quantisation.

Pruning These compression approaches remove entirely parts of the network such as weights close to zero (Gordon et al., 2020; Mao et al., 2020; Chen et al., 2020) and weights that move towards zero during fine-tuning (Sanh et al., 2020; Tambe et al., 2021). Different to operating on individual weights, previous work attempted to remove structured blocks of weights or even architectural components such as attention heads and encoder layers (Fan et al., 2019; Prasanna et al., 2020; Khetan and Karnin, 2020; Li et al., 2020a; Lin et al., 2020; Tay et al., 2021).

Knowledge Distillation This set of techniques typically train a light-weight student model to mimic the outputs of a larger teacher PLM (Sun et al., 2019; Li et al., 2020b; Jiao et al., 2020; Sun et al., 2020b; Li et al., 2021; Tahaei et al., 2022). In a similar direction, smaller PLMs have been recently fine-tuned on text generated by larger PLMs (Chiang et al., 2023; Taori et al., 2023).

Weight Matrix Decomposition Previous work also proposed replacing large weight matrices by the product of two smaller ones for reducing model size and runtime memory. Weight matrix decomposition has been applied to linear layers (Mao et al., 2020; Ben Noach and Goldberg, 2020), the embedding matrix (Lan et al., 2020b; Tambe et al., 2021; Wang et al., 2022), and attention blocks (Hu et al., 2022; Wang et al., 2022).

Embedding Matrix Compression Finally, various attempts have been introduced for compressing the embedding matrix during pre-training and fine-tuning (Xue et al., 2022; Clark et al., 2022b; Xue and Aletras, 2022).

3.2.2 Improving Attention Efficiency

Previous work on making attention more efficient includes efforts towards (1) speeding-up pairwise computations between token representations; and (2) parameter efficiency.

Computational Efficiency While improving computational efficiency of attention is out of the scope of our paper, we provide a brief overview of previous work since it is complementary to parameter efficiency. One approach to speed up attention computation is by reducing the number

of similarity computations between representations in different positions using predefined local windows, fixed or dynamic strides (Child et al., 2019; Zaheer et al., 2020; Beltagy et al., 2020; Kitaev et al., 2020). Other methods leverage the approximation of SoftMax to change the order of matrix multiplications, resulting in lower computational complexity (Katharopoulos et al., 2020; Choromanski et al., 2021; Schlag et al., 2021; Qin et al., 2022). Related approaches along this direction proposed kernel functions that require additional parameters (Choromanski et al., 2021; Wang et al., 2020). Finally, Dao et al. (2022) proposed improvements in GPU memory access to optimise and accelerate the MHA computation.

Memory Efficiency Lan et al. (2020a) introduced a method for sharing the projection parameters for queries, keys and values across transformer layers. Furthermore, Kitaev et al. (2020) proposed sharing the projection matrix between keys and values within each layer. Additionally, other methods use a multi-query attention approach that shares projection weights for keys and values across different heads (Shazeer, 2019; Chowdhery et al., 2022; Ainslie et al., 2023b), while Yan et al. (2021) directly treat the input hidden states as both keys and values. In a different direction, Lee-Thorp et al. (2022) proposed replacing the attention blocks with faster token-mixture blocks consisting of a few parameters or no parameters at all. This includes methods such as linear or Fourier transformations in the token-mixture block. However, these approaches tend to yield lower predictive performance compared to MHA.

3.3 Multiple Head Embeddings Attention

Inspired by the absolute position embeddings (Vaswani et al., 2017; Devlin et al., 2019) for distinguishing the representation of the same token in different contexts, we propose Multiple Head Embeddings (MHE) attention. MHE uses a shared 'seed' projection matrix that is subsequently combined with distinct head embeddings to generate multiple attention heads.

3.3.1 Multi-head Attention (MHA)

We first begin by formally defining MHA. MHA consists of different projection matrices ($\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V \in \mathbb{R}^{d_m \times d_h}, i = 1, ..., n$, where d_m is the dimension of the input representation and d_h is the dimension of n attention heads) for queries (Q), keys (K) and values (V) per head, $3 \times n$ in total. It is computed as follows:

$$\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i = \mathbf{X}\mathbf{W}_i^{Q,K,V} \tag{3.1}$$

$$\mathbf{H}_i = \text{Att}(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i) \tag{3.2}$$

$$= \text{SoftMax}\left(\frac{\mathbf{Q}_i\mathbf{K}_i^\top}{\sqrt{d_h}}\right)\mathbf{V}_i \tag{3.3}$$

Note that we use scale-dot attention, but our method can be used with any other attention mechanism.

3.3.2 Seed Projection Matrix

Unlike MHA that uses different projection matrices per head, MHE attention employs only a single projection matrix for each of the queries, keys and values, $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d_m \times d_h}$. These matrices are shared across all attention heads.

We obtain query, key and values projections of the input sequence $\mathbf{X}$ as follows:

$$\mathbf{Q}, \mathbf{K}, \mathbf{V} = \mathbf{X}\mathbf{W}^{Q,K,V} \tag{3.4}$$

3.3.3 Attention Head Embeddings

Using a seed projection matrix for $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ is equivalent to a single-head attention (SHA) module. Therefore, we need a mechanism to transform the seed projection matrices to obtain different attention heads. For this purpose, we represent each attention head i by specific head embeddings $\mathbf{e}_i^Q, \mathbf{e}_i^K, \mathbf{e}_i^V \in \mathbb{R}^{d_h}, i = 1, ..., n$ for queries, key and values. These embeddings have a substantially smaller memory footprint compared to using different projection matrices per head. The contextualized representation $\mathbf{H}_i$ of the entire input sequence $\mathbf{X}$ for head i is computed as follows:

$$\tilde{\mathbf{Q}}_i, \tilde{\mathbf{K}}_i, \tilde{\mathbf{V}}_i = \psi(\mathbf{Q}; \mathbf{K}; \mathbf{V}, \mathbf{e}_i^{Q,K,V}) \tag{3.5}$$

$$\mathbf{H}_i = \text{Att}(\tilde{\mathbf{Q}}_i, \tilde{\mathbf{K}}_i, \tilde{\mathbf{V}}_i) \tag{3.6}$$

where $\psi(\cdot)$ is a function that modifies the query, key and value matrices with a corresponding head embedding $\mathbf{e}_i$.

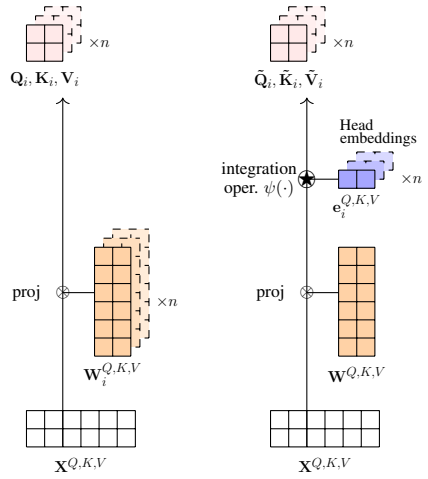
3.3.4 Modifying Queries, Keys and Values with Head Embeddings

We propose two MHE variants, one adds and the other multiplies the head embeddings with the seed projection matrices.

MHE-ADD: Motivated by the absolute position embedding (Devlin et al., 2019), we use the addition operation in Equation 3.5, represented as $\psi(\mathbf{A}, \mathbf{b}) := \mathbf{A} + \mathbf{1}\mathbf{b}^\intercal$, where $\mathbf{A} \in \{\mathbf{Q}, \mathbf{K}, \mathbf{V}\}$ and $\mathbf{b} \in \{\mathbf{e}^Q, \mathbf{e}^K, \mathbf{e}^V\}$ respectively.

MHE-MUL: Likewise, motivated by the rotary position embedding (Su et al., 2021), MHE-MUL employs multiplication as the integrating operation in Equation 3.5 as $\psi(\mathbf{A}, \mathbf{b}) := \mathbf{A} \odot \mathbf{1}(\mathbf{b} + \mathbf{1})^\intercal$, where $\odot$ represents the Hadamard product.[4]

[4]We add 1 to avoid elements in queries, keys and values become too small during initialisation.



(a) **MHA** (b) **MHE**

Figure 3.2: Multi-head attention (left) requires $3 \times n$ projection matrices for queries, keys and values ($\mathbf{W}^{Q,K,V}$) where n is the number of attention heads. Multi-head embedding attention (right) uses only three projection matrices and $3 \times n$ head embeddings.

Figure 3.2 shows an overview of the MHE mechanism compared to MHA.

3.4 Experimental Setup

3.4.1 Attention Mechanisms

We compare our MHE attention with the following attention mechanisms:[5]

- **Multi-head Attention (MHA):** This is the original multi-head attention mechanism (Vaswani et al., 2017; Devlin et al., 2019).
- **Single-head Attention (SHA):** Similar to MHA but using only one attention head.
- **EL-ATT:** Introduced by Yan et al. (2021), this attention variant completely eliminates the projection matrices for all keys and values.
- **MQA:** Introduced by Shazeer (2019), this approach uses shared projection matrices for keys and values across all attention heads. Note that different projection matrices are used for queries across heads.
- **SKV:** Introduced by Kitaev et al. (2020), this attention variant enforces keys and values to share the same projection matrix within each attention module.

[5]We have also experimented with Linear and Fourier token-mixture models (Lee-Thorp et al., 2022) yielding substantially lower performance. For full results of these methods, see section 3.8.

3.4.2 Data

We experiment with a diverse range of tasks including: (1) two standard natural language understanding benchmarks in English, GLUE (Wang et al., 2018) and SUPERGLUE (Wang et al., 2019); (2) two question and answering benchmarks in English, SQUAD V1.1 (Rajpurkar et al., 2016) and SQUAD V2.0 (Rajpurkar et al., 2018); (3) WMT-14 English-to-German machine translation (Bojar et al., 2014); and (4) two language modelling datasets in English WIKITEXT-103 (Merity et al., 2017) and PENN TREEBANK (Marcus et al., 1993).

3.4.3 Models

We test all different attention variants on two architectures: (1) encoder-only transformer (Devlin et al., 2019) and (2) encoder-decoder transformer (Vaswani et al., 2017).

Encoder-only For GLUE, SUPERGLUE, SQUAD V1.1 and SQUAD V2.0, we use a BERT-base architecture. This consists of 12 transformer layers, embedding size of 768, hidden states dimension of 768, 12 attention heads and a maximum sequence length of 512.

Decoder-only We also test a decoder-only model using the GPT2-base architecture on WIKITEXT-103, PENN TREEBANK and GLUE. GPT2-base consists of 12 transformer layers, embedding size of 768, hidden states dimension of 768, 12 attention heads and a maximum sequence length of 512.

Encoder-decoder For WMT-14, we train an encoder-decoder transformer from scratch. It consists of 12 layers (6 for the encoder and decoder respectively), an embedding size of 512, hidden states dimension of 512 and 8 attention-heads and a maximum sequence length of 100.

We set the number of attention heads to 1 for all SHA models. Experimenting with larger models and different number of attention heads is out of the scope of our paper and left for future work due to limited access to computing resources.

3.4.4 Implementation Details

Pre-training We pre-train all models on the English Wikipedia and BookCorpus (Zhu et al., 2015) from HuggingFace (Lhoest et al., 2021) for up to 1M steps with a batch size of 128. We choose masked language modelling as the pre-training objective. For all models, we use a 30K WordPiece vocabulary (Devlin et al., 2019).

Fine-tuning and Training For GLUE, SUPERGLUE, SQUAD V1.1 and SQUAD V2.0, we fine-tune all pre-trained models up to 20 epochs with early stopping fixing the batch size to 32. For each task, we use five different seeds and report the average.

		GLUE			SUPERGLUE			SQUAD V1.1			SQUAD V2.0		
Attention	**#params↓**	Acc↑	PRR↑	PEoP↑	Acc↑	PRR↑	PEoP↑	Acc↑	PRR↑	PEoP↑	Acc↑	PRR↑	PEoP↑
SHA	8.85M	79.2	96.7	-	67.1	95.1	-	82.5	93.1	-	67.6	87.8	-
MHA	28.32M	81.9	100.0	0.02	70.5	100.0	0.02	88.6	100.0	0.03	77.0	100.0	0.06
EL-ATT	14.16M	80.3	98.0	0.02	69.5	98.5	0.06	86.5	97.6	0.08	72.2	93.8	0.11
MQA	15.34M	81.3	99.2	0.04	69.3	98.2	0.04	86.7	97.9	0.07	74.8	97.1	0.15
SKV	21.23M	**81.4**	**99.4**	0.02	**69.9**	**99.1**	0.03	**88.1**	**99.4**	0.05	**75.9**	**98.6**	0.09
MHE-ADD	8.88M	80.4	98.2	4.92	69.1	97.9	9.44	83.7	94.5	4.65	71.8	93.2	19.88
MHE-MUL	8.88M	80.6	98.3	**5.53**	69.6	98.7	**12.07**	85.9	97.0	**13.19**	72.3	93.9	**22.25**

Table 3.1: Results of the encoder-only architecture on GLUE, SUPERGLUE, SQUAD V1.1 and SQUAD V2.0 dev sets with performance retention ratio (PRR) and performance elasticity of parameters (PEoP) over five runs. **Bold** values denote best performing method in each benchmark. ↑ and ↓ denote that higher and lower values are preferred, respectively.

		GLUE			WIKITEXT-103			PENN TREEBANK		
Attention	**#params↓**	Acc↑	PRR↑	PEoP↑	PPL↑	PRR↑	PEoP↑	PPL↑	PRR↑	PEoP↑
SHA	8.85M	75.3	97.2	-	62.0	55.8	-	68.1	46.3	-
MHA	28.32M	77.5	100.0	0.01	43.0	100.0	0.14	44.3	100.0	0.16
EL-ATT	14.16M	76.6	98.9	0.03	57.1	67.2	0.13	56.1	73.4	0.29
MQA	15.34M	76.9	99.2	0.03	49.7	84.4	0.27	49.3	88.7	0.38
SKV	21.23M	**77.1**	**99.5**	0.02	**46.2**	**92.6**	0.18	**45.5**	**97.3**	0.24
MHE-ADD	8.88M	75.8	97.8	2.18	54.0	74.4	41.29	55.3	75.2	60.15
MHE-MUL	8.88M	76.7	99.0	**5.92**	53.8	74.9	**42.32**	50.7	85.6	**81.76**

Table 3.2: Results of decoder-only architecture on GLUE dev sets and WIKITEXT-103, PENN TREEBANK test sets with performance retention ratio (PRR) and performance elasticity of parameters (PEoP) over five runs. **Bold** values denote best performing method in each benchmark. ↑ and ↓ denote that higher and lower values are preferred, respectively.

We train the encoder-decoder model from scratch on the training set of WMT-14 English-to-German machine translation dataset up to 100K steps with a batch size of 256. WMT-14 contains 4.5M sentence pairs and evaluate on its test set. We train the tokeniser using byte-pair-encoding (Sennrich et al., 2016) with 37K merging steps on the training set. We enable both source language and target language to share the vocabulary. We use one random seed and report the average on the last five epochs. We optimise all models using AdamW (Loshchilov and Hutter, 2019).

Hyperparameters Hyperparameter selection details are in section 3.9.

Hardware For pre-training, we use four NVIDIA Tesla A100 GPUs and one for fine-tuning on downstream tasks.

3.4.5 Predictive Performance Evaluation

For GLUE, SUPERGLUE, SQUAD V1.1 and SQUAD V2.0, we use the official metric of each task (see section 3.8 for details on metrics for each task). We report F1 score for SQUAD V1.1 and SQUAD V2.0. We use BLEU to report performance in WMT-14 English-to-German machine translation task. We use perplexity (PPL) to report generative performance on WIKITEXT-103 and PENN TREEBANK by fixing the stride length to 256.

3.4.6 Memory Efficiency Evaluation

Furthermore, we use the following metrics to measure and compare the memory efficiency of MHE and the baselines.

- **Performance Retention Ratio:** We compute the ratio between the predictive performance of each attention mechanism compared to MHA upper-bound baseline performance (the higher the better).

  For direct indicator (e.g. accuracy etc.):

  $$\text{PRR} = \frac{\text{score}_{\text{model}}}{\text{score}_{\text{MHA}}}$$

  For inverse indicator (e.g. perplexity etc.):

  $$\text{PRR} = 1 - \frac{\text{score}_{\text{model}} - \text{score}_{\text{MHA}}}{\text{score}_{\text{MHA}}}$$

- **Performance Elasticity of Parameters:** Inspired by the concept of elasticity in economics (Bittermann, 1934), which measures the responsiveness of an economic variable (e.g. investment demand) to a change in another (e.g. interest rate), we extend it to measure the parameter utilisation rate of a target model compared to the SHA lower-bound. The performance elasticity of parameters (PEoP) indicates how effectively parameters contribute to predictive performance, compared to SHA. It is computed as the ratio between the gradient of predictive performance score and the gradient of parameter counts at the point of SHA lower-bound, detailed in the following:

  For direct indicator (e.g. accuracy etc.):

  $$\text{PEoP} = \frac{(\text{score}_{\text{model}}/\text{score}_{\text{SHA}}) - 1}{(\text{params}_{\text{model}}/\text{params}_{\text{SHA}}) - 1}$$

  For inverse indicator (e.g. perplexity etc.):

  $$\text{PEoP} = -\frac{(\text{score}_{\text{model}}/\text{score}_{\text{SHA}}) - 1}{(\text{params}_{\text{model}}/\text{params}_{\text{SHA}}) - 1}$$

PEoP quantifies the extent to which a model's performance can be boosted with 1% additional parameters compared to a baseline model (the higher the better).[6]

3.5 Results

3.5.1 Predictive Performance Comparison

Table 3.1 presents results on GLUE, SUPERGLUE, SQUAD V1.1 and SQUAD V2.0 for our MHE variants and all baselines. We first observe that both the performance of our MHE-ADD and MHE-MUL are comparable to the vanilla MHA on two text classification benchmarks (80.4, 80.6 vs. 81.9 on average GLUE and 69.1, 69.6 vs. 70.5 on average SUPERGLUE) with high performance retention ratios (PRR) between 97.9% and 98.7%. On question answering tasks SQUAD V1.1 and SQUAD V2.0, both MHE variants are also competitive, with PRRs higher than 93%.

Similar results are observed on the WMT-14 English-to-German machine translation task for the encoder-decoder transformer. According to Table 3.3, MHE-ADD and MHE-MUL achieve BLEU scores of 23.0 and 23.6, respectively. The performance of MHE-MUL is negligibly lower than that of MHA (24.8) while being substantially smaller.

Consistent results for the decoder-only transformer are shown in Table 3.2. The PRRs for MHE-ADD and MHE-MUL on GLUE are still high (i.e. 97.8% and 99.0%). While using the intrinsic metrics for evaluation, MHE-MUL leads to the perplexities of 53.8 and 50.7 compared to 43.0 and 44.3 for MHA on WIKITEXT-103 and PENN TREEBANK respectively, indicating a stable PRR higher than 74.9%.

In all tasks, MHE consistently outperforms SHA by a large margin with only 0.03M extra parameters, i.e. 0.6~17.4. For example, 69.6 vs. 67.1 in SUPERGLUE, 72.3 vs. 67.6 in SQUAD V2.0, 23.6 vs. 22.5 in WMT-14 and 62.0 vs. 53.8 in WIKITEXT-103 for the MHE-MUL variant. We also note that MQA and SKV attention mechanisms generally perform better than MHE, however they are 1.7 and 2.4 times larger than MHE, i.e. 15.34M and 21.23M vs. 8.88M parameters. It is worth noting that MHE-MUL outperforms EL-ATT on three out of five benchmarks, despite having nearly half the parameters in the attention module.

3.5.2 Memory Efficiency Comparison

Our results so far indicate that performance increases with the number of attention mechanism parameters, which is expected. Next, we inspect how efficiently different attention mechanisms utilise their parameters. [7] Tables 3.1 and 3.3 show how parameter efficient our two MHE

[6] We subtract 1 in both nominator and denominator, following the original definition of elasticity.

[7] For a detailed report on the memory usage of different attention mechanisms, see Appendix 3.10.

attention variants and all baselines are, measured in PEoP. Note that PEoP scores for SHA cannot be computed as it is used as the point for reference model. We also report PRR using MHA as a baseline for completeness, however this metric does not take the model size into account.

We first observe in Table 3.1 that both our MHE-ADD and MHE-MUL achieve the highest PEoP scores on the two natural language understanding benchmarks (4.92, 5.53 on GLUE, and 9.44, 12.07 on SUPERGLUE) and two question answering tasks (4.65, 13.19on SQUAD V1.1, and 19.88, 22.25 on SQUAD V2.0). In contrast, vanilla MHA results in the lowest PEoP score among all models as expected, ranging from 0.02 to 0.06. It indicates the memory inefficiency of MHA.

The PEoPs of more light-weight EL-ATT and SKV are similar to that of MHA (0.02) on average GLUE, barely 4 ‰of that of MHE, indicating they are far more memory-inefficient compared to MHE.

Similar findings are observed in WMT-14 for the encoder-decoder models depicted in Table 3.3. MHE-ADD and MHE-MUL achieve PEoP scores of 20.0 and 27.9, respectively. In contrast, the PEoP scores of MHA, EL-ATT MQA and SKV are close to zero (barely 0.1). This means that investing more parameters into their attention modules would not bring proportional benefits in predictive performance. Even for the SKV which is half the size of MHA and achieves high PRR, when the number of parameters increase by 1%, the BLEU score increases a negligible 0.1%, while evolving from SHA. However, with the same number of parameters, our most memory-inefficient MHE-MUL is able to improve the BLEU score by 11.0%. Such rate of return is 110 times larger than that of SKV. Leveraging the head embeddings by adding only a negligible number of parameters efficiently improves the predictive performance.

We further observe that MHE-ADD and MHE-MUL are architecture-agnostic, obtaining similar memory efficiency for the decoder-only model in Table 3.2. Both our MHE-ADD and MHE-MUL achieve the highest PEoP scores on the two language modelling benchmarks (41.29, 42.32 on WIKITEXT-103 and 60.15 and 81.76 on PENN TREEBANK) and GLUE (2.18 and 5.92). At the same time, MHA fail to perform well on GLUE and PENN TREEBANK with a PEoP of 0.01 and 0.16 respectively. MHE-ADD and MHE-MUL also consistently outperform other efficient-attention variants (i.e. EL-ATT, MQA and SKV) by 72~340 times on PEoP across the three benchmarks.

In all tasks, MHE consistently outperforms MHA by orders of magnitude in parameter efficiency. We also note that EL-ATT, MQA and SKV only lead to PEoP scores with the same magnitude as MHA. This highlights the more superior parameter utilisation of MHE attention variants, achieving state-of-the-art memory-efficiency.

Attention	#params↓	BLEU↑	PRR↑	PEoP↑
SHA	6.49M	22.5	90.8	-
MHA	18.87M	24.8	100.0	0.1
EL-ATT	9.44M	23.9	96.6	0.1
MQA	10.62M	24.2	97.6	0.1
SKV	14.16M	**24.7**	**99.5**	0.1
MHE-ADD	6.52M	23.0	92.9	5.5
MHE-MUL	6.52M	23.6	95.0	**11.0**

Table 3.3: BLEU scores on WMT-14 English to German machine translation task with performance retention ratio (PRR) and performance elasticity of parameters (PEoP). **Bold** values denote best performing method in each benchmark. ↑ and ↓ denote that higher and lower values are preferred, respectively.

3.5.3 Theoretical Memory Complexity

Table 3.4 presents the theoretical memory complexity and the total number of parameters of our two MHE and baseline attention mechanisms in a single transformer sublayer. First, we see that the theoretical memory complexity of MHA and other efficient parameters (EL-ATT, MQA and SKV) are quadratic with the number of attention heads, while our MHE are the only two variants having the complexity linear with the attention heads similar to SHA.

Taking a closer look at the rightmost column in Table 3.4, we observe that the number of extra parameters of all attention variants compared to SHA have a quadratic relationship to both the number n and the dimension of attention heads d, except our two MHE variants. MHE only requires a relatively small fraction of additional parameters compared to SHA.

3.5.4 Scaling the Number of Attention Parameters

Delving deeper to the effect of scaling to memory footprint, we show in Figure 3.3 the total number of parameters needed for a single attention module (e.g. in an encoder layer). We fix the dimension of attention heads to 64 commonly used by BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), GPT-2 (Radford et al., 2019), BART (Lewis et al., 2020) and T5 (Raffel et al., 2020). In general, we note that the number of parameters in MHA could reach more than 200M if employing 128 attention heads. At the same time, SKV, MQA and EL-ATT would require 2/3, 1/3 and 1/3 of that number respectively. In contrast, MHE only accounts for 1% of the MHA parameters.

Moreover, we also present in Figure 3.4 the total number of parameters required across attention

Attention	Complexity↓	#Params↓	#Params (+)↓
SHA	$\mathcal{O}(n)$	$3d^2n$	0
MHA	$\mathcal{O}(n^2)$	$3d^2n^2$	$(3n^2-3n)d^2$
EL-ATT	$\mathcal{O}(n^2)$	d^2n^2	$(n^2-3n)d^2$
MQA	$\mathcal{O}(n^2)$	$d^2n^2+2d^2n$	$(n^2-n)d^2$
SKV	$\mathcal{O}(n^2)$	$2d^2n^2$	$(2n^2-3n)d^2$
MHE (ours)			
-ADD	$\mathcal{O}(n)$	$3d^2n+3dn$	$3nd$
-MUL	$\mathcal{O}(n)$	$3d^2n+3dn$	$3nd$

Table 3.4: Memory complexity regarding the number of parameters in each attention sublayer, while fixing the dimension of attention heads to d. n denotes the number of attention heads. To simplify, the dimension of hidden states d_m is set to nd. The last projection for pooling attention heads is excluded. ↑ and ↓ denote that higher and lower values are preferred, respectively.

variants when stacking 12, 24 and 48 layers along with 32 and 64 attention heads respectively. We also fix the dimension of attention heads to 64. We can observe, when the number of attention head reaches 64, MHA with 24 layers already occupies more than 1B parameters, while EL-ATT and MQA reach 0.8B parameters with 48 layers. SKV takes 24 layers to reach 0.8B parameters. However, the total number of parameters in MHE attention does not exceed 0.1B even when scaling to 48 layers with 64 attention heads. It is also clear that scaling the attention module to 48 layers, 32 attention heads and 12 layers needs a comparable number of parameters for MHA, EL-ATT, MQA or SKV. This indicates, that LLM developers have to make a choice whether doubling the number of attention heads or cutting down the number of layers to a quarter when working under a tight memory budget. However, MHE does not suffer by such issues.

Further, we project these estimates to the popular GPT-3 model (Brown et al., 2020). It is a decoder-only model with 96 decoder layers, 96 attention heads per layer, and a head dimension of 128. The vanilla multi-head attention module requires a massive 43.48B parameters. However, using MHE attention, this number can be significantly reduced to 0.46B parameters, i.e. approximately a reduction by 98.9%.[8] Comparing this to other parameter-efficient attention variants such as EL-ATT (14.50B parameters), MQA attention (14.80B parameters), and SKV attention (28.99B parameters), it becomes evident that our MHE offers better memory efficiency. This makes it a compelling alternative for memory-constrained scenarios. See Appendix 3.11

[8]It would have been great to report results by pre-training our own MHE GPT-3 model, however this is prohibitive with the modest compute we have available.

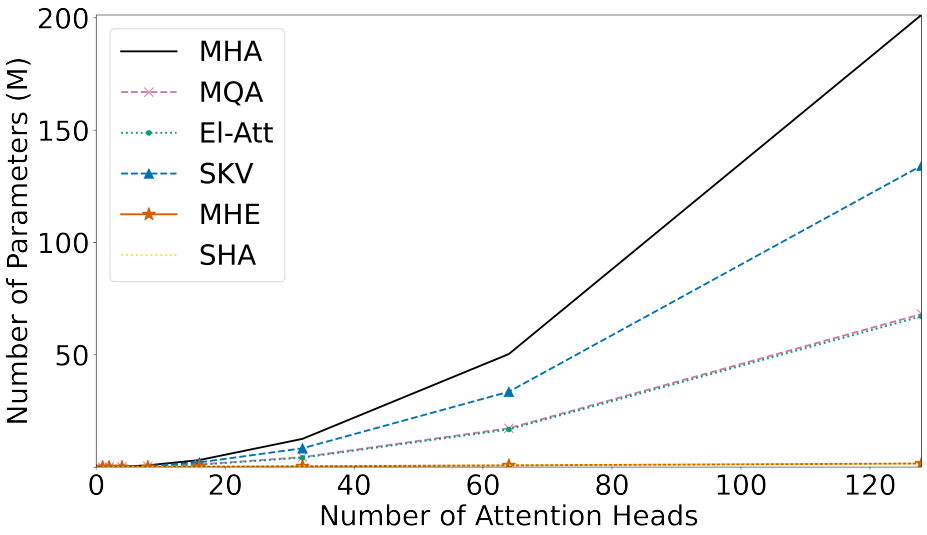


Figure 3.3: Number of parameters per attention sublayer, while scaling the number of attention heads in different attention variants. We fix the dimension of attention to 64.

for a detailed study on the robustness of MHE to model size changes (i.e. scaling).

3.6 Discussion

MHA enables the model to attend to information from different representation subspaces at different positions (Vaswani et al., 2017). It uses distinct projection matrices for each attention head and integrates the information from these different representation subspaces. However, Vaswani et al. (2017) did not explore different methods for performing space transformations per head.

Previous work has pointed out that over-parameterized models might have a low intrinsic dimension. Therefore, transforming the projection matrices to smaller low-rank ones usually does not severely harm model predictive performance (Li et al., 2018; Aghajanyan et al., 2020). Meanwhile, the classic MHA approach also does not impose any constraints on the orthogonality of these subspaces during pre-training and fine-tuning. The column vectors in those projection matrices could be highly collinear, i.e. the projection matrices could be rank-deficient. As a result, its inner-working mechanism could be simply understood as introducing levels of variation to the encoded representation of the same token at the same position across different heads.

Our MHE approach is possible to achieve memory efficiency (similar to SHA) together with high PRR compared to MHA by mimicking the position embeddings for representing different

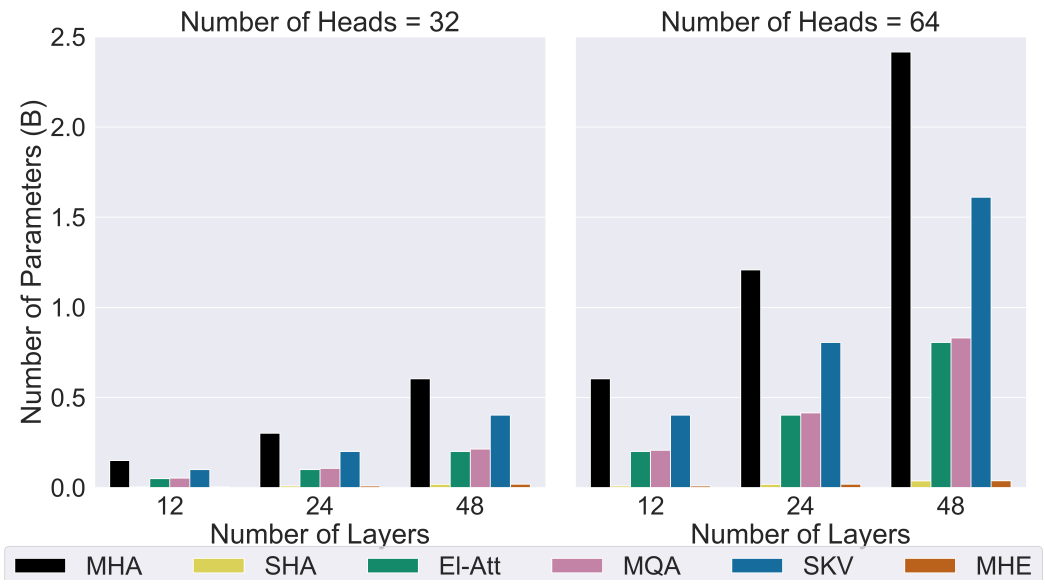


Figure 3.4: Total number of parameters in attention sublayers, while scaling the number of attention layers to 12, 24 and 48 with 32 attention heads and 64 attention heads respectively. We fix the dimension of attention to 64.

attention heads.

On one hand, the addition operation in MHE-ADD is used for transforming the keys, queries and values. This can be seen as a small distortion of the subspace obtained through projection, followed by rotation. For an input representation, the difference between the projected and injected (i.e. through head embedding addition) queries, keys and values is a constant vector across any pair of heads. On the other hand, the MHE-MUL approach employs a multiplication operation, which more aggressively distorts and reshapes the keys, queries and values subspaces. Head embeddings in MHE-MUL play a role as the scaling factors, respectively stretching each dimension of the input representation. Thus, the difference between the keys, queries, and values generated by different heads for the same input representation, is a vector parallel to the projected input. This vector is dependent on the specific input, unlike the constant vector in MHE-ADD.

Interestingly, our experimental results consistently show that the multiplication operation outperforms addition in the majority of benchmarks. This corroborates findings of a previous empirical study by Su et al. (2021) that compared rotary position embeddings (somehow analogous to MHE-MUL) with absolute position embeddings (analogous to MHE-ADD).

3.7 Conclusions

We have proposed MHE attention that employs a single shared projection matrix along with multiple head embeddings, to simplify and reduce the memory footprint of the MHA. Our experimental results have demonstrated that MHE attention exhibits superior memory efficiency compared to other memory-efficient attention variants, while achieving high predictive performance ratio to MHA on various downstream tasks. Compared to a single-head attention, MHA requires $(3n^2 - 3n)d^2$ parameters for n attention heads and head dimensionality d, while MHE barely requires a negligible $3nd$. For future research, we plan to investigate scaling up MHE models and explore its linguistic capabilities (Vulić et al., 2020; Koto et al., 2021).

Limitations

We experiment only using 'base' size models without experimenting with larger architectures, due to limited access to computational resources. Similarly, we did not experiment with decoder only architectures (Brown et al., 2020) which we leave for future work. We have not combined our MHE method with computationally efficient attention methods with linear complexity, such as Linformer (Wang et al., 2020). We expect that it would speed up computation of MHE, but it is out of the scope of our paper.

Acknowledgments

We would like to thank Constantinos Karouzos, Miles Williams and the anonymous reviewers for their invaluable feedback.

ATTENTION	MNLI↑	QNLI↑	QQP↑	RTE↑	SST↑	MRPC↑	CoLA↑	STS↑	GLUE Avg.↑
SHA	80.5(0.3)	87.5(0.2)	86.7(0.1)	63.6(0.9)	90.7(0.3)	85.1(0.7)	53.8(1.1)	85.8(0.4)	79.2(0.1)
MHA	83.4(0.1)	89.8(0.3)	87.8(0.1)	67.6(1.5)	92.0(0.3)	86.8(0.4)	59.6(1.3)	88.5(0.3)	81.9(0.3)
EL-ATT	81.7(0.1)	88.4(0.2)	87.3(0.2)	67.6(1.0)	**91.7**(0.6)	85.9(0.7)	52.4(1.7)	87.7(0.2)	80.3(0.3)
MQA	**82.6**(0.1)	88.8(0.2)	87.3(0.1)	66.5(0.9)	91.4(0.5)	87.3(0.2)	**58.4**(1.3)	87.9(0.2)	81.3(0.2)
SKV	**82.6**(0.1)	**89.4**(0.3)	**87.7**(0.1)	**68.2**(1.7)	91.6(0.3)	**87.4**(0.6)	56.2(1.2)	**88.6**(0.2)	**81.4**(0.2)
FNET	76.3(0.1)	83.8(0.1)	84.8(0.1)	63.2(2.0)	88.4(0.7)	78.0(0.4)	43.2(2.5)	83.7(0.3)	75.2(0.6)
LINEAR	75.4(0.1)	81.4(0.1)	85.5(0.2)	54.7(2.3)	90.4(0.4)	72.2(0.6)	50.3(1.0)	70.9(0.5)	72.6(1.1)
MHE-ADD	81.5(0.2)	87.8(0.2)	87.2(0.1)	66.9(2.0)	90.5(0.4)	87.2(0.3)	54.7(0.9)	87.7(0.1)	80.4(0.2)
MHE-MUL	81.9(0.1)	87.9(0.1)	87.4(0.1)	67.1(1.5)	91.1(0.5)	85.4(0.5)	56.6(1.7)	87.3(0.2)	80.6(0.2)
MHA(M)	84.4(0.2)	91.1(0.4)	84.0(0.6)	70.5(1.0)	92.0(0.2)	87.2(0.8)	62.5(1.0)	88.8(0.2)	82.6(0.4)
MHE-MUL (M)	82.7(0.2)	89.2(0.4)	87.2(0.2)	67.9(0.4)	90.7(0.3)	86.3(1.0)	59.8(1.8)	88.0(0.2)	81.5(0.3)

Table 3.5: Results for encoder-only models on GLUE dev sets with standard deviations over five runs in parentheses. **Bold** values denote best performing method in each task. ↑ and ↓ denote that higher and lower values are preferred, respectively.

3.8 Appendix A: Reported Metrics for Each Task

We evaluate all models on GLUE (Wang et al., 2018), SUPERGLUE (Wang et al., 2019), SQUAD V1.1 (Rajpurkar et al., 2016) and SQUAD V2.0 (Rajpurkar et al., 2018). We report matched accuracy for MNLI, Matthews correlation for CoLA, Spearman correlation for STS, F1 score for QQP, CB, MultiRC and SQUAD and accuracy for all other tasks. Table 3.5 and Table 3.6 present results on GLUE and SUPERGLUE respectively for our MHE-FORMERS models and all baselines with the encoder-only architecture. Table 3.7 and 3.8 present results of the scores and performance elasticity of parameters (PEoP) across all models over each task in GLUE and SUPERGLUE. Table 3.10 presents results on GLUE for our MHE-FORMERS models and all baselines with the decoder-only architecture. Table 3.10 presents results of the scores and performance elasticity of parameters (PEoP) across all models over each task in GLUE.

3.9 Appendix B: Hyperparameters

The hyperparameters used in pre-training are listed in Table 3.11. The hyperparameters used in fine-tuning are listed in Table 3.12.

3.10 Appendix C: Memory Usage

To further illustrate the memory-efficiency of our MHE models compared to the baselines, we take the BERT-base architecture (12 attention heads, each with a dimension of 64) as an example, and measure the memory usage per attention block as in Section 2.1.1 from Smith

ATTENTION	BoolQ↑	CB↑	RTE↑	WiC↑	MultiRC↑	COPA↑	WSC↑	SUPERGLUE Avg.↑
SHA	72.3(0.7)	88.7(2.5)	62.5(1.0)	63.6(0.5)	59.7(15.7)	59.2(2.8)	63.8(1.5)	67.1(2.4)
MHA	76.6(0.6)	89.4(1.8)	67.9(1.3)	65.4(0.8)	69.0(1.2)	64.0(3.1)	61.5(3.3)	70.5(0.5)
EL-ATT	73.5(1.0)	85.7(4.8)	**69.5**(1.4)	63.8(0.8)	67.9(0.3)	62.2(1.8)	63.8(0.5)	69.5(1.0)
MQA	74.6(0.6)	86.7(1.6)	65.4(0.8)	64.0(1.2)	**68.8**(0.5)	62.2(2.0)	63.3(2.7)	69.3(0.7)
SKV	**75.2**(0.3)	84.5(3.7)	67.5(0.9)	**65.2**(1.1)	68.7(0.2)	**64.0**(1.0)	**64.4**(1.2)	**69.9**(0.4)
FNET	68.4(0.5)	51.8(4.3)	60.7(0.9)	63.8(1.1)	62.3(0.6)	58.2(3.7)	60.0(1.6)	60.7(0.6)
LINEAR	70.4(0.2)	50.6(2.1)	55.2(1.8)	62.9(0.7)	57.8(0.5)	60.0(2.8)	61.0(1.1)	59.7(0.9)
MHE-ADD	73.3(0.2)	88.8(1.7)	67.5(1.5)	64.2(0.5)	67.1(0.2)	60.2(2.8)	62.5(1.4)	69.1(0.5)
MHE-MUL	74.9(0.6)	**89.4**(1.0)	67.8(1.3)	64.7(0.6)	68.0(0.3)	61.6(1.5)	61.2(2.9)	69.6(0.3)
MHA(M)	78.1(0.3)	88.1(6.8)	70.3(1.3)	67.8(0.8)	72.9(0.6)	68.2(4.1)	64.6(2.9)	72.9(0.6)
MHE-MUL (M)	75.2(0.5)	84.6(2.4)	68.6(1.8)	66.3(0.9)	69.8(0.4)	61.6(3.8)	64.6(0.8)	70.1(1.1)

Table 3.6: Results for encoder-only models on SUPERGLUE dev sets with standard deviations over five runs in parentheses. **Bold** values denote best performing method in each task. ↑ and ↓ denote that higher and lower values are preferred, respectively.

ATTEN	GLUE													
-TION	MNLI↑		QNLI↑		QQP↑		RTE↑		SST↑		MRPC↑		CoLA↑		STS↑	
SHA	80.5	-	87.5	-	86.7	-	63.6	-	90.7	-	85.1	-	53.8	-	85.8	-
MHA	83.4	(0.02)	89.8	(0.01)	87.8	(0.01)	67.6	(0.03)	92.0	(0.01)	86.8	(0.01)	59.6	(0.05)	88.5	(0.01)
EL-ATT	81.7	(0.02)	88.4	(0.02)	87.3	(0.01)	67.6	(0.10)	<u>91.7</u>	(0.02)	85.9	(0.02)	52.4	(-0.04)	87.7	(0.04)
MQA	<u>82.6</u>	(0.04)	88.8	(0.02)	87.3	(0.01)	66.5	(0.06)	91.4	(0.01)	87.3	(0.04)	<u>58.4</u>	(0.12)	87.9	(0.03)
SKV	<u>82.6</u>	(0.02)	<u>89.4</u>	(0.02)	<u>87.7</u>	(0.01)	<u>68.2</u>	(0.05)	91.6	(0.01)	<u>87.4</u>	(0.02)	56.2	(0.03)	<u>88.6</u>	(0.02)
FNET	76.3	(-)	83.8	(-)	84.8	(-)	63.2	(-)	88.4	(-)	78.0	(-)	43.2	(-)	83.7	(-)
LINEAR	75.4	(-)	81.4	(-)	85.5	(-)	54.7	(-)	90.4	(-)	72.2	(-)	50.3	(-)	70.9	(-)
MHE-ADD	81.5	(3.88)	87.8	(1.34)	87.2	(1.86)	66.9	(16.35)	90.5	(-0.81)	87.2	**(7.93)**	54.7	(5.22)	87.7	**(7.05)**
MHE-MUL	81.9	**(5.41)**	87.9	**(1.51)**	87.4	**(2.54)**	67.1	**(17.80)**	91.1	**(1.29)**	85.4	(1.29)	56.6	**(16.29)**	87.3	(5.60)

Table 3.7: Detailed average scores and performance elasticity of parameters (in parentheses) on GLUE for MHE models and the baselines with encoder-only architecture using MLM as pre-training objectives. **Underlined** values denote the best performing method and **bold** values denote the method with best PEoP in each task. ↑ and ↓ denote that higher and lower values are preferred, respectively.

et al. (2022) and report the memory usage saving ratio (%) during the attention calculation in Table 3.13:

The calculation is based on inputs with batch size of 32, hidden dimension of 768, sequence length of 512 and fp16 mixture precision training using the following formula:

- Memory(weights)=#params*(2+4) bytes;

ATTEN -TION	SuperGlue BoolQ↑	CB↑	RTE↑	WIC↑	MultiRC↑	COPA↑	WSC↑
SHA	72.3 -	88.7 -	62.5 -	63.6 -	59.7 -	59.2 -	63.8 -
MHA	76.6 (0.03)	89.4 (0.00)	67.9 (0.04)	65.4 (0.01)	69.0 (0.07)	64.0 (0.04)	61.5 (-0.02)
EL-ATT	73.5 (0.03)	85.7 (-0.06)	69.5 (0.19)	63.8 (0.00)	67.9 (0.23)	62.2 (0.08)	63.8 (0.00)
MQA	74.6 (0.04)	86.7 (-0.03)	65.4 (0.06)	64.0 (0.01)	68.8 (0.21)	62.2 (0.07)	63.3 (-0.01)
SKV	75.2 (0.03)	84.5 (-0.03)	67.5 (0.06)	65.2 (0.02)	68.7 (0.11)	64.0 (0.06)	64.4 **(0.01)**
FNET	68.4 (-)	51.8 (-)	60.7 (-)	63.8 (-)	62.3 (-)	58.2 (-)	60.0 (-)
LINEAR	70.4 (-)	50.6 (-)	55.2 (-)	62.9 (-)	57.8 (-)	60.0 (-)	61.0 (-)
MHE-ADD	73.3 (4.58)	88.8 (0.54)	67.5 (25.50)	64.2 (3.00)	67.1 (39.90)	60.2 (5.41)	62.5 (-6.75)
MHE-MUL	74.9 **(11.78)**	89.4 **(2.52)**	67.8 **(26.97)**	64.7 **(5.36)**	68.0 **(44.63)**	61.6 **(12.97)**	61.2(-13.49)

Table 3.8: Detailed average scores and performance elasticity of parameters (in parentheses) on SUPERGLUE for MHE models and the baselines with encoder-only architecture using MLM as pre-training objectives. **Underlined** values denote the best performing method and **bold** values denote the method with best PEoP in each task. ↑ and ↓ denote that higher and lower values are preferred, respectively.

ATTENTION	MNLI↑	QNLI↑	QQP↑	RTE↑	SST↑	MRPC↑	CoLA↑	STS↑	GLUE Avg.↑
SHA	78.7(0.1)	86.0(0.2)	85.0(0.1)	66.5(0.9)	89.8(0.2)	76.8(0.4)	38.0(1.3)	81.5(0.4)	75.3(0.3)
MHA	80.6(0.1)	87.9(0.2)	86.3(0.1)	66.9(1.1)	90.2(0.3)	79.0(0.7)	42.9(1.3)	86.0(0.2)	77.5(0.2)
EL-ATT	79.5(0.2)	86.8(0.3)	85.7(0.1)	65.7(1.4)	90.0(0.4)	79.2(1.4)	41.5(2.2)	84.3(0.2)	76.6(0.4)
MQA	80.0(0.1)	86.3(0.1)	**85.9**(0.1)	66.2(0.7)	90.3(0.3)	**80.7**(0.6)	41.3(0.8)	84.3(0.4)	76.9(0.2)
SKV	**80.3**(0.1)	**87.5**(0.3)	**85.9**(0.1)	66.1(1.1)	90.6(0.5)	79.6(0.5)	**41.9**(1.8)	**84.9**(0.2)	**77.1**(0.4)
MHE-ADD	78.7(0.1)	85.6(0.2)	85.4(0.1)	66.4(2.5)	89.6(0.4)	78.7(0.6)	38.4(1.2)	83.5(0.3)	75.8(0.3)
MHE-MUL	79.0(0.2)	85.5(0.1)	85.6(0.1)	**70.2**(2.5)	**90.9**(0.2)	78.9(0.8)	39.4(1.3)	84.0(0.3)	76.7(0.2)

Table 3.9: Results for decoder-only models on GLUE dev sets with standard deviations over five runs in parentheses. **Bold** values denote best performing method in each task. ↑ and ↓ denote that higher and lower values are preferred, respectively.

- Memory(gradients)=#params*(2+4) bytes;
- Memory(Adam states)=#params*(4+4) bytes;
- Memory(activations)= batch-size*sequence-length*hidden-dimension*2 bytes.

From Table 3.13, we observe the memory usage saving ratio of our proposed MHE is 2.75 times better than SKV, 1.50 times better than MQA and 1.37 times better than EL-ATT, which indicates a SotA memory saving capabilities compared to all other parameter-efficient attention variants.

ATTEN -TION	GLUE MNLI↑	QNLI↑	QQP↑	RTE↑	SST↑	MRPC↑	CoLA↑	STS↑
SHA	78.7 -	86.0 -	85.0 -	66.5 -	89.8 -	76.8 -	38.0 -	81.5 -
MHA	80.6 (0.01)	87.9 (0.01)	86.3 (0.01)	66.9 (0.00)	90.2 (0.00)	79.0 (0.01)	42.9 (0.06)	86.0 (0.03)
EL-ATT	79.5 (0.02)	86.8 **(0.02)**	85.7 (0.01)	65.7 (-0.02)	90.0 (0.01)	79.2 (0.05)	41.5 (0.16)	84.3 (0.06)
MQA	80.0 (0.02)	86.3 (0.01)	85.9 (0.01)	66.2 (-0.01)	90.3 (0.01)	80.7 (0.07)	41.3 (0.12)	84.3 (0.05)
SKV	80.3 (0.01)	87.5 (0.01)	85.9 (0.01)	66.1 (-0.00)	90.6 (0.01)	79.6 (0.03)	41.9 (0.07)	84.9 (0.03)
MHE-ADD	78.7 (-0.02)	85.6 (-1.50)	85.4 (1.60)	66.4 (-0.69)	89.6 (-0.57)	78.7 (7.96)	38.4 (3.71)	83.5 (7.98)
MHE-MUL	79.0 **(0.98)**	85.5 (-1.88)	85.6 **(2.05)**	70.2 **(17.72)**	90.9 **(4.01)**	78.9 **(8.58)**	39.4 **(12.32)**	84.0 **(9.97)**

Table 3.10: Detailed average scores and performance elasticity of parameters (in parentheses) on GLUE for MHE models and the baselines with decoder-only architecture using MLM as pre-training objectives. **Underlined** values denote the best performing method and **bold** values denote the method with best PEoP in each task. ↑ and ↓ denote that higher and lower values are preferred, respectively.

Hyperparameter	Pretraining
Maximum train steps	1000000 steps
Batch size (per GPU)	32 instances
Adam ϵ	1e-8
Adam β_1	0.9
Adam β_2	0.9999
Sequence length	512
Peak learning rate	1e-4 for MLM
Learning rate schedule	linear
Warmup steps	10000
Weight decay	0.01
Attention Dropout	0.1
Dropout	0.1

Table 3.11: Details of hyperparameters used in pre-training.

3.11 Appendix D: Robustness to Scaling

We also conduct experiments to observe the effectiveness and the robustness of our best MHE-MUL while scaling the model size.

Table 3.14 presents average accuracy on two text classification benchmarks (GLUE and SUPERGLUE), perplexities on two language modelling benchmarks (WIKITEXT-103 and PENN TREEBANK) with their corresponding performance retention ratio (PRR) for MHA and MHE-MUL in both encoder-only and decoder-only architecture across different model sizes.[9] For

[9]BASE: 12 encoder/decoder layers, each containing 12 attention heads; LARGE/MEDIUM: 24 encoder/decoder

Hyperparameter	Fine-tuning
Maximum train epochs	20 epochs for GLUE, SUPERGLUE and SQUAD
Batch size (per GPU)	32 instances
Adam ϵ	1e-6
Adam β_1	0.9
Adam β_2	0.999
Peak learning rate	3e-5 for GLUE and SQUAD; 5e-5 for SUPERGLUE
Learning rate schedule	cosine with hard restarts
Warmup steps	first 6% steps for GLUE and SUPERGLUE; 3327 for SQUAD V1.1; 4950 for SQUAD V2.0
Weight decay	0
Attention Dropout	0.1
Dropout	0.1
Evaluation steps	2455 for MNLI, 655 for QNLI, 2275 for QQP, 48 for RTE, 421 for SST, 69 for MRPC, 162 for CoLA and 108 for STS, 177 for BoolQ, 5 for CB, 47 for RTE, 102 for WiC, 512 for MultiRC, 8 for COPA, 11 for WSC, 548 for SQUAD V1.1, 815 for SQUAD V2.0

Table 3.12: Details of hyperparameters used in fine-tuning.

ATTENTION	**weights**↓	**gradients**↓	**Adam states**↓	**activations**↓	**Total**↓	**Memory Saving Ratios** (%)↑
SHA	4423680	4423680	5898240	25165824	39911424	44.84
MHA	14155776	14155776	18874368	25165824	72351744	0.00
EL-ATT	7077888	7077888	9437184	25165824	48758784	32.61
MQA	7667712	7667712	10223616	25165824	50724864	29.89
SKV	10616832	10616832	14155776	25165824	60555264	16.30
MHE-ADD	4437504	4437504	5916672	25165824	39957504	44.77
MHE-MUL	4437504	4437504	5916672	25165824	39957504	44.77

Table 3.13: Memory usage (in bytes) and memory saving ratios (compared to MHA) per attention block for our MHE and other baselines. MHA denotes BERT-base here. ↑ and ↓ denote that higher and lower values are preferred, respectively.

layers, each containing 16 attention heads.

		#Params(M)↓		GLUE↑			SUPERGLUE↑			WIKITEXT-103↓			PENN TREEBANK↓		
		MHA	MHE -MUL	MHA	MHE -MUL	PRR↑	MHA	MHE -MUL	PRR↑	MHA	MHE -MUL	PRR↑	MHA	MHE -MUL	PRR↑
Encoder	BASE	28.32	8.88	81.9	80.6	98.4	70.5	69.6	98.7	-	-	-	-	-	-
-only	LARGE	100.66	29.96	81.5	82.6	98.7	72.9	70.1	96.2	-	-	-	-	-	-
Decoder	BASE	28.32	8.88	77.5	76.7	99.0	-	-	-	43.0	53.8	74.9	44.3	50.7	85.6
-only	MEDIUM	100.66	29.96	79.4	77.7	97.9	-	-	-	35.5	37.2	95.2	37.5	41.6	88.5

Table 3.14: Results of evaluation metrics on two text classification benchmarks (GLUE, SUPERGLUE) and two language modelling benchmarks (WIKITEXT-103 and PENN TREEBANK) with performance retention ratio (PRR) for MHA and MHE-MUL across different model sizes. ↑ and ↓ denote that higher and lower values are preferred, respectively.

the encoder-only models, we observe that the PRR of MHE-MUL remains stable on GLUE (from 98.4% to 98.7%) and SUPERGLUE (from 98.7% to 96.2%) while scaling the number of parameters in the attention blocks to 3.5 times larger. For the decoder-only models, the PRR on GLUE for MHE-MUL stabilises at 97.9% (i.e. 1.1% lower) after scaling. Surprisingly, the PRR of MHE-MUL increases on WIKITEXT-103 (from 74.9% to 95.2%) and PENN TREEBANK (from 85.6% to 88.5%) while scaling to MEDIUM size.

Similar results are observed for the encoder-decoder architecture on WMT14 machine translation task. According to Table 3.15, we first observe the PRR of MHE-MUL remains stable (i.e. between 91.5% and 96.0%) across all different sizes, where the number of parameters in the corresponding MHA ranges from 19.87M to 75.50M. Meanwhile, we also observe that making the model deeper by stacking more encoder and decoder layers results in a steady increment on PRR for MHE-MUL (e.g. 93.6%, 95.0% and 96.0% respectively, for 8 layers, 12 layers and 16 layers in total). Moreover, for the same number of parameters in the attention, wider attention heads consistently leads to a better PRR for MHE-MUL, i.e. 91.5%, 95.0% and 95.3% for the dimensionality of 32, 64 and 128 of attention heads respectively.

These results indicate MHE consistently achieves good performance retention ratios and is robust to model size change.

	$\mathbf{N}$	$\mathbf{d_m}$	$\mathbf{h}$	$\mathbf{d_h}$	$\mathbf{p_{drop}}$	**#Steps**	**#Params**(M)↓		**BLEU**↑		**PRR**↑
							MHA	MHE-MUL	MHA	MHE-MUL	
BASE	12	512	8	64	0.1	100K	18.87	6.52	24.8	23.6	95.0
	12	512	16	32	0.1	100K	18.87	5.63	25.1	22.9	91.5
	12	512	4	128	0.1	100K	18.87	8.29	24.7	23.6	95.3
4L	8	512	8	64	0.1	100K	12.58	4.34	23.9	22.4	93.6
8L	16	512	8	64	0.1	100K	25.17	8.69	25.3	24.3	96.0
12H	12	768	12	64	0.15	100K	42.47	13.31	25.7	24.2	94.2
BIG	12	1024	16	64	0.3	300K	75.50	22.47	26.5	24.8	93.6

Table 3.15: Results of BLEU scores on WMT-14 English to German machine translation task with performance retention ratio (PRR) for MHA and MHE-MUL across different model sizes. ↑ and ↓ denote that higher and lower values are preferred, respectively.

Chapter 4

Publication III:

Deconstructing Attention: Investigating Design Principles for Effective Language Modeling

The primary contribution in this chapter is the under-review paper, *Deconstructing Attention: Investigating Design Principles for Effective Language Modeling*. The project began with a question about the viability of using a single vector for state tracking and a shared softmax partition function to create a simple linear attention model. Initial experiments on this include straightforward linear attention, exploring augmented approaches such as selective space, more complex gating mechanisms, and even expanding to multiple latent vectors. However, none of these methods outperformed the standard dot-product attention. Other attempts are made to simplify the attention mechanism, including pre-caching attention scores or even adding augmented methods like weighting edges through predicted magnitude. However, variants along this line still not perform well on down-stream tasks. Encouraged by the motto of The University of Sheffield[1][2].

[1] Felix, qui potuit rerum cognoscere causas. –*Virgil (70 – 19 BC)*

[2] Thro' known Effects can trace the secret Cause. –*The works of Virgil, 1697*

The coat of arms of the University of Sheffield above the gate of Sir Frederick Mappin Building, photographed by Huiyin on August 23, 2025.

Abstract

The success of Transformer language models is widely credited to their dot-product attention mechanism, which interweaves a set of key design principles: mixing information across positions (enabling multi-token interactions), sequence-dependent activations (where attention weights adapt to each input), a specific mathematical form (dot-product similarities plus softmax weighting), and coupling of queries and keys to evolving hidden states (grounding attention in the current layer). However, the necessity of each of these principles remains largely untested. In this work, we systematically deconstruct attention by designing controlled variants that selectively relax these principles, applied both uniformly across all layers and in hybrid architectures where only some layers retain standard attention. Our empirical analysis reveals that mechanisms for mixing tokens are indispensable, as their absence collapses models to near-random behavior, while the exact mathematical form and sequence dependency can be substantially relaxed, especially when preserved in just a subset of layers. Surprisingly, even variants that fail in isolation can achieve robust performance when interleaved with standard attention, highlighting a cooperative effect. These findings deepen our understanding of what truly underpins attention's effectiveness and open new avenues for simplifying language models without sacrificing performance.[3]

[3]Code is available at https://anonymous.4open.science/r/DeconAttn-FB16.

4.1 Introduction

The remarkable success of Transformer-based language models (Singh, 2025; Liu et al., 2024a; Yang et al., 2024a, LMs) is widely attributed to the dot-product attention mechanism (i.e. standard attention), which enables these models to weight the significance of each token in a sequence by computing pairwise similarities of their contextual representations (Vaswani et al., 2017). However, this powerful mechanism comes at a substantial computational cost with respect to the input sequence length (L). This has led to a diverse landscape of proposed mechanisms, including processing longer context (Tay et al., 2022a), token-mixing via pooling and multi-layer perceptron MLP-Mixer (Tolstikhin et al., 2021), non-parametric transformations (Yu et al., 2022; Lee-Thorp et al., 2022), optimized kernel functions (Aksenov et al., 2024; Arora et al., 2024; Qin et al., 2022; Peng et al., 2021; Kasai et al., 2021; Choromanski et al., 2021; Katharopoulos et al., 2020), and linear recurrent neural network (RNN) architectures (Siems et al., 2025; Peng et al., 2025; Dao and Gu, 2024; Yang et al., 2024b; Qin et al., 2024; Peng et al., 2024; Poli et al., 2023; Peng et al., 2023; Orvieto et al., 2023).

Despite this rich body of work, most of these approaches implicitly preserve several underlying design principles inherited from standard attention. Broadly, these principles include: (1) incorporating mechanisms for mixing information across tokens (`Token Mixing`), enabling multi-token interactions, (2) emulating the original mathematical form of standard attention (`Mathematical Form`), i.e. dot-product similarities followed by softmax weighting, (3) enforcing strict sequence-dependency in activation maps (`Sequence-Dependency`), where attention weights depend on the specific input sequence, and (4) deriving queries and keys from the current layer's hidden states (`Current QK`), as opposed to other input types such as uncontextualized representations. However, the importance of each of these principles remains largely untested. *Are all of these truly essential, or could relaxing some of them suffice if applied selectively?*

Motivated by this foundational question and guided by Occam's Razor (Baker, 2022), we take a diagnostic approach: we systematically relax these principles through controlled attention variants, evaluated in two settings: (1) uniform replacement across all layers, and (2) hybrid configurations that interleave standard and simplified modules. Through extensive empirical analysis across multiple model sizes, attention variants, and layer configurations, while carefully matching parameter counts of variants, we uncover a set of insights that refine our understanding of key attention principles.

Under *uniform* replacement, mechanisms enabling token mixing prove indispensable: removing them, e.g. in *MLP* variants, leads to near-random accuracy on challenging natural language understanding (NLU) tasks, though such models still capture superficial statistical patterns, as reflected in improved perplexity over trivial baselines. Retaining the dot-product structure

and sequence-dependent weighting contributes to stability, but these elements are not strictly necessary in every layer, provided token interactions remain strong.

Notably, in hybrid configurations that interleave simpler attention mechanisms with standard layers, we uncover a striking pattern: attention variants that fail in isolation can nonetheless contribute meaningfully when paired with standard attention, achieving robust performance that often matches or exceeds fully standard models. This suggests standard layers may stabilise activations, mitigate distributional drift, and foster cooperative dynamics across the network, as reflected in both predictive outcomes and structural diagnostics such as attention entropy, head diversity, and sink behaviors.

While hybrid attention schemes have been explored in prior work, such as taking advantages of state space models (Glorioso et al., 2024) or augmenting feed-forward modules via mixture-of-experts routing (Lenz et al., 2025), these are typically driven by performance or efficiency goals. *By contrast, our hybrid designs serve as deliberate probes to isolate and examine the causal roles of specific attention properties.* Taken together, our findings challenge the assumption that attention mechanisms must adhere rigidly to their original formulation. By identifying which components are essential and which can be simplified, we outline a path toward new LM architectures that can be structurally leaner and adaptable.

4.2 Related Work

Prior research attributes the success of Transformer models to their efficient token mixing mechanisms. Consequently, numerous studies explore replacing the standard dot-product attention with simpler architectural components that enable parallel training. For instance, Yu et al. (2022) demonstrate the effectiveness of pooling, MLPs, and convolution as alternatives within vision Transformers. Similarly, Lee-Thorp et al. (2022) highlight the efficiency of token mixers based on Fourier transformation and random projection in the BERT model (Devlin et al., 2019). However, these investigations focus on encoder-only Transformer architectures and may not readily adapt to causal language modeling. While Tolstikhin et al. (2021) introduce a learnable linear layer for token mixing by employing position-wise projection vectors, similar to Linformer (Wang et al., 2020), this approach encounters scalability challenges with long sequences due to its parameter count growing linearly with L. Concurrent research largely retains the standard dot-product attention mechanism as a foundational principle. Efforts to reduce the computational cost of this mechanism primarily follow two strategies: weight sharing (Rajabzadeh et al., 2024; Ainslie et al., 2023a; Xue and Aletras, 2023; Yan et al., 2021; Kitaev et al., 2020; Shazeer, 2019; Xiao et al., 2019) or input length shrinkage (Nawrot et al., 2023; Clark et al., 2022b; Xue and Aletras, 2022).

Recent work revisits linear RNNs to handle inputs of varying length (Gu and Dao, 2023; Poli et al., 2023; Peng et al., 2023; Orvieto et al., 2023; Gu et al., 2022). Follow-up research further improves performance by designing more sophisticated gating mechanisms and update rules (He et al., 2025; Lin et al., 2025; Siems et al., 2025; Peng et al., 2025; Dao and Gu, 2024; Yang et al., 2024b; Qin et al., 2024; Peng et al., 2024), with the goal of mimicking human memory, drawing inspiration from the work of Schlag et al. (2021) on fast weight programmers. Notably, such replacements can also be selectively applied to a subset of attention layers or heads (Lenz et al., 2025; Ren et al., 2025; Team et al., 2024; Glorioso et al., 2024; Peng and Cao, 2024; Dong et al., 2024; Tay et al., 2019). Additionally, this work operates on the contextual representations encoded by deep networks to generate activation maps dynamically.

Another line of research approximates the dot-product computation to achieve linear complexity. These methods rely on various kernel functions that emulate the exponential function using its Taylor expansion (Aksenov et al., 2024; Arora et al., 2024; Qin et al., 2022; Peng et al., 2021; Kasai et al., 2021; Choromanski et al., 2021; Katharopoulos et al., 2020). This allows for prioritisation of the key-value dot product through feature mapping. However, this line of work does not examine the necessity of the other key principles of attention mechanism identified in §4.1.

4.3 Attention Variants

To operationalise our investigation of the four key design principles identified in §4.1, we design targeted variations of attention that selectively relax each property. This allows us to probe their necessity in a controlled, principled framework.

4.3.1 Standard Dot-product Attention

We take standard scaled dot-product attention (Vaswani et al., 2017) as our baseline, where queries ($\mathbf{Q}$), keys ($\mathbf{K}$), and values ($\mathbf{V}$) are computed from the layer hidden states $\mathbf{H} \in \mathbb{R}^{L \times d_m}$:

$$\mathbf{O} = \mathrm{Att}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{A}\mathbf{V} \tag{4.1}$$

$$\mathbf{A} = \mathrm{Softmax}\left(\mathbf{Q}\mathbf{K}^\top / \sqrt{d_h}\right) \tag{4.2}$$

$$\mathbf{Q}, \mathbf{K}, \mathbf{V} = \mathbf{H}\mathbf{W}^{Q,K,V} \tag{4.3}$$

This follows all principles: mixing information across positions via $\mathbf{A}$, using a similarity-softmax form, adapting to each input sequence, and tying $\mathbf{Q}$, $\mathbf{K}$ to the current hidden state $\mathbf{H}$.

4.3.2 Relaxing Token Mixing

MLP. To directly examine the necessity of cross-token interactions, we replace attention with a gated MLP layer, consisting of three fully-connected (FC) layers ($\mathrm{FC}_{\mathrm{Dn}}, \mathrm{FC}_{\mathrm{Gt}}, \mathrm{FC}_{\mathrm{Up}}$) for down-projection, gating and down-projection respectively. This effectively eliminates token mixing and each token is processed independently, only attending to itself.

$$\mathbf{O} = \mathrm{GatedMLP}(\mathbf{H}) \tag{4.4}$$

$$= \mathrm{FC}_{\mathrm{Dn}}(\mathrm{SiLU}(\mathrm{FC}_{\mathrm{Gt}}(\mathbf{H})) \cdot \mathrm{FC}_{\mathrm{Up}}(\mathbf{H})) \tag{4.5}$$

We use a SiLU activation (Elfwing et al., 2018) and match the parameter count of standard attention. This variant serves as a minimal baseline to assess how much attention's effectiveness depends on cross-token interaction, beyond what feed-forward paths alone can provide without using any $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$.

4.3.3 Relaxing the Mathematical Form

We assess whether attention must strictly follow the canonical dot-product plus softmax formulation. To this end, we evaluate two variants that either approximate or break this form.

Approximate. Following Arora et al. (2024), we preserve the mathematical intention of similarity-based weighting, while relaxing the exact form of softmax via a second-order Taylor expansion, yielding a linear-time recurrent form (section 4.16):

$$\mathbf{A} \approx \mathrm{Taylor}\left(\mathbf{Q}\mathbf{K}^\top / \sqrt{d_h}\right) \tag{4.6}$$

$\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ are computed using Eq. 4.3.

Non-approximate. To contrast this, we introduce a new variant that discards explicit pairwise similarity altogether. Instead of computing an attention matrix via $\mathbf{Q}\mathbf{K}^\top$, it uses element-wise self-gating, multiplying $\mathbf{Q}$ and $\mathbf{K}$ derived from the same hidden state, and normalises the result across time steps with softmax:

$$\mathbf{A} = \mathrm{Softmax}\left((\mathbf{Q} \odot \mathbf{K})\mathbf{1}^\top / \sqrt{d_h}\right) \tag{4.7}$$

$$\mathbf{Q} = \mathrm{SiLU}\left(\mathbf{H}\mathbf{W}^{Q}\right); \quad \mathbf{K}, \mathbf{V} = \mathbf{H}\mathbf{W}^{K,V} \tag{4.8}$$

This variant follows an entirely different mathematical form to standard attention. We expect that this should make it harder for adjacent context tokens to receive large attention scores, as the denominator in the softmax computation monotonically increases (see recurrent form in section 4.16). Notably, the SiLU activation is applied element-wise and does not introduce

additional complexity. We apply SiLU activation on $\mathbf{Q}$ projection to add non-linearity. This does not require additional parameters and allows parallelism during training.

4.3.4 Relaxing Sequence Dependency

To test whether attention weights must be dynamically adapted to each input sequence (i.e. sequence-dependent), we construct two variants where $\mathbf{Q}$ and $\mathbf{K}$ are fixed across all inputs, inspired by MLP-Mixer (Tolstikhin et al., 2021), but making the parameter count in attention blocks independent of the maximum sequence length. Relaxing sequence dependency allows attention scores for all inputs to be pre-computed and cached during inference.

Random-fixed (RndEmbQK). We initialise a set of random embeddings $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2\mathbf{I})$ that remain constant across inputs. These are passed through the Transformer stack up to layer l:

$$\mathbf{X} = \text{TransformerBlock}^{(l)}(\epsilon), \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \sigma\mathbf{I}) \tag{4.9}$$

$$\mathbf{Q}, \mathbf{K} = \mathbf{X}\mathbf{W}^{Q,K}; \quad \mathbf{V} = \mathbf{H}\mathbf{W}^{V} \tag{4.10}$$

Since $\mathbf{Q}$ and $\mathbf{K}$ do not depend on the input, attention maps are fixed and do not adapt to context.

Text-fixed (FixedSeqQK). Instead of random embeddings, we use a randomly selected fixed sequence of natural language tokens $\mathbf{t}^s$ (first 2048 tokens from FineWeb-10BT (Lozhkov et al., 2024)). These are embedded and passed through the Transformer to generate $\mathbf{X}$:

$$\mathbf{X} = \text{TransformerBlock}^{(l)}(\text{Emb}(\mathbf{t}^s)) \tag{4.11}$$

$$\mathbf{Q}, \mathbf{K} = \mathbf{X}\mathbf{W}^{Q,K}; \quad \mathbf{V} = \mathbf{H}\mathbf{W}^{V} \tag{4.12}$$

This setup also produces fixed attention maps, but grounded in natural text instead of completely randomly initialised embeddings. Compared to *RndEmbQK*, it may encode weak structural priors, such as grammatical patterns or token co-occurrences. These variants allow us to test whether dynamic, input-conditioned attention maps are necessary, or whether fixed maps, paired with learned value paths, are sufficient.

4.3.5 Relaxing the Derivation of Q and K

StaticEmbQK. Finally, to test whether tying $\mathbf{Q}, \mathbf{K}$ to current layer hidden states ($\mathbf{H}$ or $\mathbf{X}$ above) is essential, we compute them directly from static input embeddings $\mathbf{e}$ corresponding to the input sequence $\mathbf{t}$:

$$\mathbf{Q}, \mathbf{K} = \mathbf{e}\mathbf{W}^{Q,K}; \quad \mathbf{V} = \mathbf{H}\mathbf{W}^{V}; \quad \mathbf{e} = \text{Emb}(\mathbf{t}) \tag{4.13}$$

This means that while attention maps are not fixed, they are computed without contextualisation from the evolving hidden representations. It further allows attention scores from different layers to be computed in parallel.

4.4 Experimental Setup

4.4.1 Data

We use seven zero-shot NLU tasks in English: ARC-E (Clark et al., 2018), BoolQ (Clark et al., 2019), COPA (Roemmele et al., 2011), PiQA (Bisk et al., 2020), SciQ (Welbl et al., 2017), RTE (Wang et al., 2019) and HellaSwag (Zellers et al., 2019). We also experiment with two LM tasks: WikiText (Merity et al., 2017) and Lambada Openai (Radford et al., 2019).

4.4.2 Implementation Details

Base model. Our models are built upon Qwen2.5 (Yang et al., 2024a). However, we replace its standard attention mechanism with the alternative attention modules detailed in § 4.3. To ensure a strict parameter count match across all attention variants, we use multi-head attention (Vaswani et al., 2017), deviating from Qwen2.5's default grouped-query attention (Ainslie et al., 2023a). For tokenisation, we use the 50K English-centric BPE (Sennrich et al., 2016) vocabulary of Pythia (Biderman et al., 2023), offering small memory footprint, and fast training.

Model configurations. We pretrain models with approximately 500M parameters, using two configurations: (1) *Uniform* with simple attention mechanisms across all Transformer layers; (2) *Hybrid* that integrates simple attention mechanisms in odd-numbered layers and standard attention in even-numbered layers. To assess the contribution of the modified attention variants within the *hybrid* configuration, we introduce a configuration where we remove the odd-numbered layers from pre-trained *hybrid* models (*skip*) and evaluate the resulting performance without additional training.

We further test these three configurations by training models of 70 million and 160 million parameters (see section 4.8). We finally explore various alternative *hybrid* configurations such as changing the simple attention replacement ratio, the details of which are presented in § 4.6. Specific model size details are provided in section 4.19. Meanwhile, we strictly constrain all models with different attention variants to have the same number of parameters to eliminate any effects from differences in size.

Pre-training. All models are pre-trained on the SlimPajama dataset (Soboleva et al., 2023) for up to 15 billion tokens, following Chinchilla scaling laws (Hoffmann et al., 2022). We use a

mini-batch size of 500K tokens, aligning with the training budget outlined in Titans (Behrouz et al., 2024). To optimise pre-training efficiency, we use a sequence length of 2048 tokens.[4]

4.4.3 Predictive Performance Evaluation

We use the LM-evaluation-harness toolkit v0.4.8 (Gao et al., 2024) for evaluation. We report accuracy for all NLU tasks and perplexity (PPL) for LM tasks. For LAMBADA OPENAI, we report both.

4.4.4 Attention Pattern Indicators

Looking at the performance itself may not offer a comprehensive picture of the behaviour of the different attention mechanisms we test. To obtain a more granular understanding of their internal workings, we investigate their attention patterns. We compute eight indicators from the attention matrices $\mathbf{A}_j \in \mathbb{R}^{L\times L}$ for each head $j = 1, \ldots, n_h$ in a given layer. We specifically focus on *attention sinks*, i.e. over-attending to the initial token in a sequence, and *local patterns* within attention matrices, i.e. prioritising nearby tokens, following prior work (Xiao et al., 2024; Han et al., 2024).[5]

Entropy (H). Measures the randomness of attention scores. Higher ENTROPY indicates more uniform attention distribution across tokens, similar to mean-pooling: $\mathrm{H} = -\sum_{a\in\mathbf{A}} a \cdot \log(a)$.

Concentration (Conc). Measures the concentration of attention. A higher Frobenius norm $\|\mathbf{A}\|_F$ indicates attention is focused on a limited number of tokens: $\mathrm{Conc} = \|\mathbf{A}\|_F = \sqrt{\sum_{a\in\mathbf{A}} a^2}$.

Head diversity (HeadDiv). Quantifies the variability of attention patterns across different heads. Calculated as the average position-wise standard deviation across heads, higher HEADDIV suggests better use of the multi-head mechanism.

$$\mathrm{HeadDiv} = \frac{2}{L(1+L)} \sum \mathrm{std}(\{\mathbf{A}_1, \ldots, \mathbf{A}_{n_h}\})$$

[4] Details on hyperparameter selection is provided in section 4.17. For both pre-training and evaluation, we use a single AMD Instinct MI300X accelerator.

[5] ENTROPY (H), CONC, and HEADDIV are min-max normalized. SINK and LOCFOCN use absolute values (LOCFOCN is scaled by two for visibility). High ENTROPY and low CONC suggest mean-pooling like behaviour. High CONC and low ENTROPY indicate focus on a few tokens. Further examination of SINK and LOCFOCN clarifies if this focus is on the first token or local tokens. Low ENTROPY and high CONC with low scores elsewhere (except HEADDIV) may point to sparse attention on mid-sequence tokens.

		ARC-E	BoolQ	COPA	PiQA	SciQ	RTE	HellaSwag	Avg.	Wiki	LAMBADA	
		acc↑	acc↑	acc↑	acc↑	acc↑	acc↑	acc↑	acc↑	ppl↓	ppl↓	acc↑
	Rnd. Guess	$25.0_{0.0}$	$50.0_{0.0}$	$50.0_{0.0}$	$50.0_{0.0}$	$25.0_{0.0}$	$50.0_{0.0}$	$25.0_{0.0}$	39.9	3E+5	3E+6	$0.0_{0.0}$
	Majority	$25.7_{0.0}$	$62.2_{0.0}$	$56.0_{0.0}$	$50.5_{0.0}$	$25.0_{0.0}$	$52.7_{0.0}$	$25.0_{0.0}$	39.9	-	-	-
	Standard	$41.5_{1.0}$	$56.6_{0.9}$	$63.0_{4.9}$	$60.9_{1.1}$	$60.2_{1.5}$	$53.1_{3.0}$	$28.3_{0.4}$	51.9	**38.1**	134.1	$22.9_{0.5}$
UNIFORM	MLP	$28.5_{0.9}$	$37.8_{0.8}$	$54.0_{5.0}$	$54.8_{1.2}$	$25.9_{1.4}$	$52.7_{3.0}$	$26.1_{0.4}$	40.0	993.5	1E+5	$0.0_{0.0}$
	Approx.	$40.7_{1.0}$	$51.5_{0.9}$	$64.0_{4.8}$	$59.9_{1.1}$	$55.0_{1.6}$	$52.3_{3.0}$	$28.1_{0.4}$	50.2	47.9	238.6	$18.5_{0.5}$
	Non-apx.	$26.8_{0.9}$	$37.8_{0.8}$	$60.0_{4.9}$	$53.2_{1.2}$	$19.3_{1.2}$	$52.3_{3.0}$	$26.0_{0.4}$	39.3	9E+4	2E+6	$0.0_{0.0}$
	RndEmbQK	$39.5_{1.0}$	$55.3_{0.9}$	$57.0_{5.0}$	$59.8_{1.1}$	$46.4_{1.6}$	$50.9_{3.0}$	$27.2_{0.4}$	48.0	84.8	6402.4	$1.3_{0.2}$
	FixedSeqQK	$39.4_{1.0}$	**59.0**$_{0.9}$	$61.0_{4.9}$	$59.4_{1.1}$	$51.2_{1.6}$	$52.7_{3.0}$	$27.5_{0.4}$	50.0	79.1	19578.1	$1.4_{0.2}$
	StaticEmbQK	$39.6_{1.0}$	$52.9_{0.9}$	$63.0_{4.9}$	$59.4_{1.1}$	$49.2_{1.6}$	$54.2_{3.0}$	$27.2_{0.4}$	49.4	79.9	2287.4	$3.3_{0.2}$
HYBRID	MLP	$37.5_{1.0}$	$49.8_{0.9}$	$60.0_{4.9}$	$60.2_{1.1}$	$54.3_{1.6}$	$52.7_{3.0}$	$26.1_{0.4}$	48.7	45.8	228.7	$20.8_{0.6}$
	Approx.	$39.9_{1.0}$	$51.5_{0.9}$	**67.0**$_{4.7}$	$60.4_{1.1}$	$60.5_{1.5}$	$53.4_{3.0}$	$28.4_{0.4}$	51.6	39.4	140.0	$23.7_{0.6}$
	Non-apx.	**42.3**$_{1.0}$	$56.8_{0.9}$	$63.0_{4.9}$	$61.7_{1.1}$	**63.0**$_{1.5}$	$54.9_{3.0}$	**28.5**$_{0.5}$	**52.9**	39.4	**133.1**	**23.8**$_{0.6}$
	RndEmbQK	$40.1_{1.0}$	$48.3_{0.9}$	$61.0_{4.9}$	$61.2_{1.1}$	$60.0_{1.5}$	$50.9_{3.0}$	$27.2_{0.4}$	49.8	39.3	157.5	$22.0_{0.6}$
	FixedSeqQK	$40.5_{1.0}$	$58.5_{0.9}$	$64.0_{4.8}$	**61.9**$_{1.1}$	$62.0_{1.5}$	$52.7_{3.0}$	$28.4_{0.4}$	52.6	38.5	354.7	$20.3_{0.6}$
	StaticEmbQK	$39.2_{1.0}$	$54.7_{0.9}$	$64.0_{4.8}$	$60.9_{1.1}$	$58.4_{1.6}$	**57.4**$_{3.0}$	$28.2_{0.4}$	51.8	38.7	140.7	**23.8**$_{0.6}$
HYBRID SKIP	MLP	$24.4_{0.9}$	$41.8_{0.9}$	$54.0_{5.0}$	$52.8_{1.2}$	$19.0_{1.2}$	$46.9_{1.7}$	$25.6_{0.4}$	37.8	2E+5	5E+6	$0.0_{0.0}$
	Approx.	$26.6_{0.9}$	$46.1_{0.9}$	$59.0_{4.9}$	$52.8_{1.2}$	$20.1_{1.3}$	$48.0_{3.0}$	$26.0_{0.4}$	39.8	2E+6	1E+7	$0.0_{0.0}$
	Non-apx.	$26.6_{0.9}$	$39.2_{0.9}$	$52.0_{5.0}$	$51.4_{1.2}$	$20.4_{1.3}$	$46.9_{3.0}$	$25.8_{0.4}$	37.5	5E+5	9E+6	$0.0_{0.0}$
	RndEmbQK	$27.4_{0.9}$	$37.8_{0.8}$	$58.0_{5.0}$	$53.3_{1.2}$	$21.1_{1.3}$	$52.7_{3.0}$	$26.1_{0.4}$	39.5	2E+4	3E+6	$0.0_{0.0}$
	FixedSeqQK	$27.2_{0.9}$	$39.4_{0.9}$	$59.0_{4.9}$	$52.3_{1.2}$	$22.1_{1.3}$	$48.4_{3.0}$	$25.9_{0.4}$	39.2	2E+5	5E+6	$0.0_{0.0}$
	StaticEmbQK	$25.5_{0.9}$	$43.0_{0.9}$	$57.0_{5.0}$	$53.1_{1.2}$	$22.0_{1.3}$	$51.6_{3.0}$	$25.9_{0.4}$	39.7	7E+4	5E+6	$0.0_{0.0}$

Table 4.1: Performance of *uniform*, *hybrid*, *skip* and *standard* models (500M). Purple (MLP), blue (Approx., Non-apx.), green (RndEmbQK, FixedSeqQK) and yellow (StaticEmbQK) denote variants that relax `Token Mixing`, `Mathematical Form`, `Sequence-Dependency` and `Current QK`, respectively. ↑ and ↓ denote that higher and lower values are preferred, respectively.

Attention sink (Sink). Detects focus on the first token. It is the average attention score assigned by all queries to the initial token. Higher Sink means a stronger attention sink: $\text{Sink} = \sum \mathbf{A}_{:,1}/L$.

Local Focus (LocFocN). Measures the attention focus on nearby tokens. It is the average attention score for tokens at a fixed relative distance N (here $N \in \{0, 1, 2, 3\}$). Higher LocFocN suggests stronger contribution from local context.

$$\text{LocFocN} = \sum \mathbf{A}_{L-N,L-N}/\left(L-N\right)$$

4.5 Results

Table 4.1 shows the performance of all model variants (§4.3), employing *uniform*, *hybrid*, and *skip* configurations across NLU and LM tasks. Results illuminate the role each design principle plays in effective language modeling.

Token mixing is crucial. The uniform MLP model, which lacks any cross-token interaction, performs near chance on most NLU tasks, highlighting that token mixing is essential for reasoning and understanding. Despite this, it achieves a much lower perplexity on WikiText (993.5 vs. 300K for *RndEmbQK*), indicating that even without explicit mixing, MLP can memorise or exploit local token statistics, likely unigram or bigram patterns. Introducing token mixing in a hybrid setup substantially improves NLU performance (e.g. 9.2 average accuracy points over uniform MLP), showing that mixing in part of the network can compensate to a degree. Still, the hybrid MLP variant has the highest WikiText perplexity among all hybrids, indicating that token mixing across all layers is important for fully modeling long-range dependencies.

Standard mathematical form is important in uniform. When applied uniformly, variants that retain the core structure of attention (e.g. *Approximate*, *RndEmbQK*, *FixedSeqQK* and *StaticEmbQK*) restore over 92% of the average NLU accuracy of attention. In contrast, *Non-approximate*, which discards this structure, performs close to random guess (39.3 vs. 39.9 on NLU Avg. accuracy). *Approximate* achieves the strongest results among uniform variants (8.8 higher PPL on WikiText), suggesting that preserving or closely approximating its mathematical form appears critical for maintaining predictive performance.

Sequence-dependency enhances the generalisation ability. To assess the role of sequence-dependent attention, we compare variants that retain similar architectures but differ in whether attention scores vary across inputs. *StaticEmbQK*, which preserves `Sequence-Dependency`, consistently outperforms *RndEmbQK* and *FixedSeqQK*, which use fixed attention patterns, particularly on LAMBADA OPENAI by around 2% higher accuracy. This pattern holds across both uniform and hybrid settings. Additionally, hybrid models that preserve sequence-dependency, such as *Approximate*, *StaticEmbQK*, and *Non-approximate*, tend to perform better on global-context benchmarks. These results suggest that input-specific attention contributes to better generalisation, even when other attention properties are simplified.

Current QK is not as essential as expected. *StaticEmbQK* relaxes `Current QK`. Though it does not match the PPL of *standard* across language modeling tasks, it results in PPL of 79.9 twice as high as 38.1 of *standard* under *uniform* configuration on WIKITEXT. It also greatly

outperforms *MLP*, reducing PPL tenfold (from 993.5 on WIKITEXT), while its predictive performance is comparable to *standard*. Moreover, under *hybrid* configuration, it achieves predictive performance comparable to *standard* baseline across all tasks. It indicates `Current QK` is not as essential for strong predictive performance as initially believed.

Layer collaboration matters. All *hybrid* models where simple attention variants are used in odd layers and standard attention in even layers achieve predictive performance comparable to *Standard* attention on both NLU and language modeling tasks. Surprisingly, *Non-approximate* attention, the worst performer in the uniform configuration, demonstrates strong performance in this *hybrid* setup, slightly surpassing *Standard* on average NLU accuracy (+1.8%) and LAMBADA OPENAI accuracy (+0.9%), while reducing PPL by 1.0. The *hybrid* configuration also alleviates the relatively higher uncertainty observed with *RndEmbQK* and *FixedSeqQK*, halving their WIKITEXT PPL by incorporating standard layers that aid in grounding attention to individual inputs. These findings suggest that layers exhibiting poor performance in isolation can be effective when combined with stronger layers (i.e. standard attention).

Considering the residual connections, which facilitate information flow along a shortcut pathway bypassing the simple attention alternatives, we further conduct an ablation study to constrain information flow solely through these residual connections. This involves skipping the non-*Standard* layers when pre-training `hybrid` models (denoted as `SKIP` in Table 4.1). The results provide further support to the assumption of layer collaboration. All variants in `w/ SKIP` perform even slightly worse than random guessing (i.e. average accuracy lower than 39.9 on NLU) and further result in PPL explosion in language modeling compared to `hybrid` by a margin. This indicates that the non-*Standard* layers, despite their simplicity or poor performance in uniform configurations, contribute positively to the overall predictive performance in hybrid architectures.

4.6 Analysis and Discussion

Attention variants. *Non-approximate* attention that relaxes standard attention's `Mathematical Form` appears to be the most challenging to train in a *uniform* configuration. Radar plots in Figure 4.1 show very low ENTROPY alongside high CONC and HEADDIV, indicating that most heads place almost all probability mass on a narrow set of mid-sequence tokens. This behaviour might stem from its monotonically increasing denominators (seeEq. 4.18). This could make it progressively harder for later tokens in the sequence to attract attention, thereby hindering effective training in uniform configurations. *StaticEmbQK* relaxing `Current QK` coupling, generally presents active token mixing from Layer 7, however, its mid-layers exhibit high similarity. Its reliance on static embeddings for attention computation limits its adaptability to

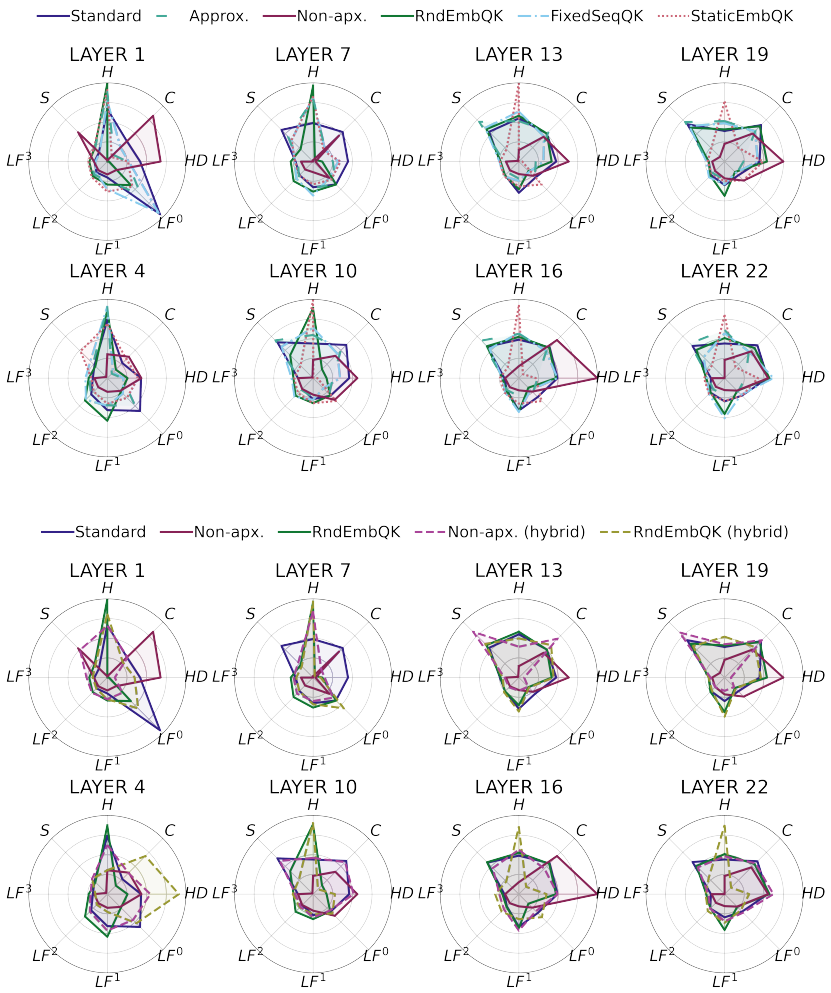


Figure 4.1: Layer-wise attention indicators for *Approx.*, *Non-approx.*, *RndEmbQK*, *FixedSeqQK* and *StaticEmbQK* in *uniform* (top) and *hybrid* (bottom) configurations, and *Standard* (H: ENTROPY, C: CONC, HD: HEADDIV, LF: LOCFOCN, S: SINK).

individual layers, further constraining predictive performance. *Approximate* and *FixedSeqQK*, showing attention patterns most similar to *Standard* across all layers. However, the performance of *FixedSeqQK* generally lags behind *Approximate*. This is due to *FixedSeqQK*'s derivation of **Q** and **K** matrices from a fixed, pre-defined text sequence, which remains constant for all inputs. Consequently, the model might become prone to simulating this specific text sequence, thereby compromising its generalisation ability. *RndEmbQK* attention faces a similar issue to *FixedSeqQK*, but suffers additional marginal performance drops, perhaps due to its inability to encode syntactic information.

Configurations. To illustrate the impact of different configurations, Figure 4.1 shows the attention patterns of *RndEmbQK* and *Non-approximate* variants as representative methods for

studying the behaviour of different attention variants in *uniform* and *hybrid* configurations (see Figure 4.8 for all layers).

With *uniform RndEmbQK* (and *uniform Standard*), the top-most layers (e.g. layer 22) exhibit high concentration (low ENTROPY and high CONC). This indicates a probability mass predominated by few selective tokens. In the hybrid design, those same layers become less selective (higher ENTROPY, lower CONC), leading to a decreased SINK score, suggesting that the hybrid mix alleviates first-token 'sink' effects. In the *Non-approximate* hybrid model, odd layers keep the *Non-approximate* heads while even layers revert to *Standard*. A clear division of labour emerges: even (*Standard*) layers mirror the baseline, balancing token mixing and focus, while odd (*Non-approximate*) layers specialise, either acting as attention sinks (high SINK, low ENTROPY) or as mean-poolers (high ENTROPY, low CONC). This complementary interplay compensates for the lower expressiveness of *Non-approximate* heads observed in the uniform setting, explaining why the *hybrid* configuration trains successfully while the *uniform* one does not.

Why hybrid works. We investigate the magnitude of raw activations (logits before softmax) within each *RndEmbQK* and *Non-approximate* layer in the *hybrid* configuration (Figure 4.2). Our analysis reveals that activations generally exhibit lower magnitudes compared to the *uniform* configuration for both attention variants. Notably, the *uniform Non-approximate* model shows activation outliers exceeding 10^3 in the final Transformer layers (e.g. Layer 21). In contrast, the *hybrid* configuration maintains activations below 10^1. This suggests that the *Standard* layers in the *hybrid* architecture might serve as a normalisation mechanism. This normalisation could mitigate over-concentration and the formation of highly sparse attention matrices, which can arise from large magnitude outliers during the numerically stable softmax operation. This normalising effect appears sufficiently strong to rescue models that are otherwise challenging to train and prone to gradient vanishing (e.g. *Non-approximate* in the *uniform* configuration).

Theoretical analysis. Li et al. (2024a) connects Transformer LMs to spin glass models. They suggest standard attention matrices align with the Gibbs-Boltzmann distribution (Gibbs, 1902), implying an implicit energy minimisation process with tokens as spins. Input-independent $\mathbf{Q}$ and $\mathbf{K}$ or form deviations disrupt this. This perspective provides a theoretical basis for the performance variations observed in our uniform replacement experiments. While Zhang et al. (2022) suggests full-rank attention offers maximal flexibility, causal attention can be low-rank due to stable softmax allowing zeros in diagonals with activation outliers. This supports our normalisation analysis in *hybrid* configurations, with Neyshabur et al. (2017)'s observation on unbalanced network training difficulty.

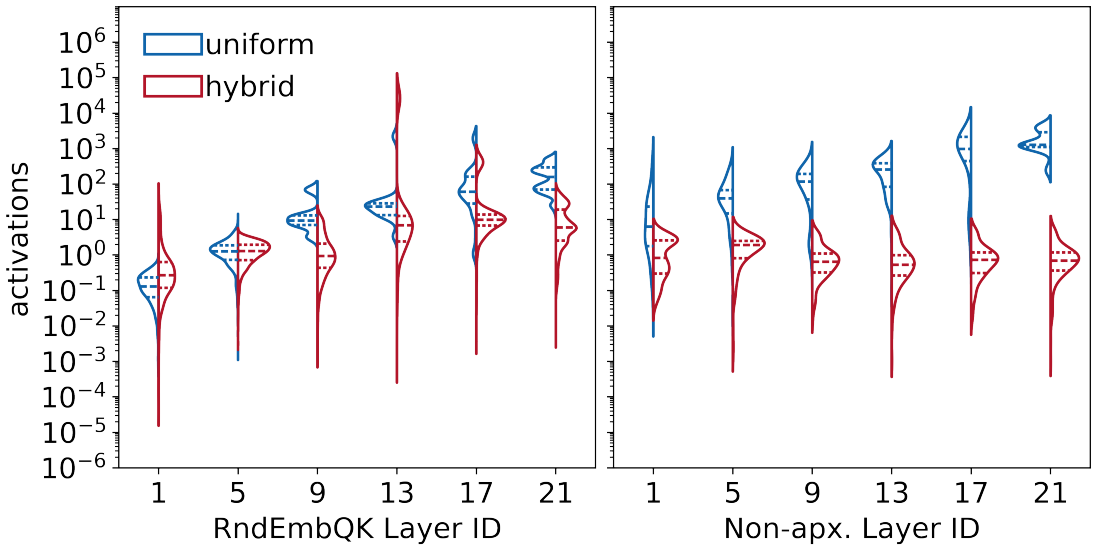


Figure 4.2: Distribution of pre-softmax activations for *RndEmbQK* (left) and *Non-approximate* (right) across two different configurations. See Figure 4.6 for all layers.

Model size. We also evaluate all attention variants across models of 70M, 160M, and 500M parameters. Our main observations remain consistent across these different model sizes. See section 4.8 for detailed results.

Hybrid configuration ablation. To investigate the impact of replacing subsets of layers with simpler attention mechanisms, we consider nine different configurations. These focus on different segments of a 24-layer architecture of the 500M model: (1) *even* or *50%* configuration, where even-numbered layers retain standard attention while odd-numbered layers are replaced; (2) *odd* configuration, with the reverse arrangement; (3) *top* configuration, where the upper layers (13-24) employ the simpler attention mechanism; (4) *middle* configuration, targeting the middle layers (7-18); (5) *bottom* configuration, focusing on the initial layers (1-6); (6) *25%*, replacing layers except Layer 4,8,12,16,20,24 with simpler attention; (7) *first*, replacing all layers with simpler attention except the first layer; (8) *last*, replacing all layers with simpler attention except the last layer; (9) *bilateral*, replacing all layers with simpler attention except Layer 1 and 24. See Table 4.11 in section 4.23 for details.

Figure 4.3 presents the predictive performance using these nine settings. For both *RndEmbQK* and *Non-approximate* mechanisms, the difference in performance across these hybrid configurations is marginal (e.g. all with a PPL around 40.0 on WIKITEXT). However, this observation does not generalise to extreme settings, such as employing *Standard* attention in only the first or the last layer. For *RndEmbQK* attention, the predictive performance remains comparable to *Standard* if only the last layer (or layers at both ends) uses *Standard*. Nevertheless, its accuracy on LAMBADA OPENAI drops to zero in such extreme cases. For *Non-approximate* attention,

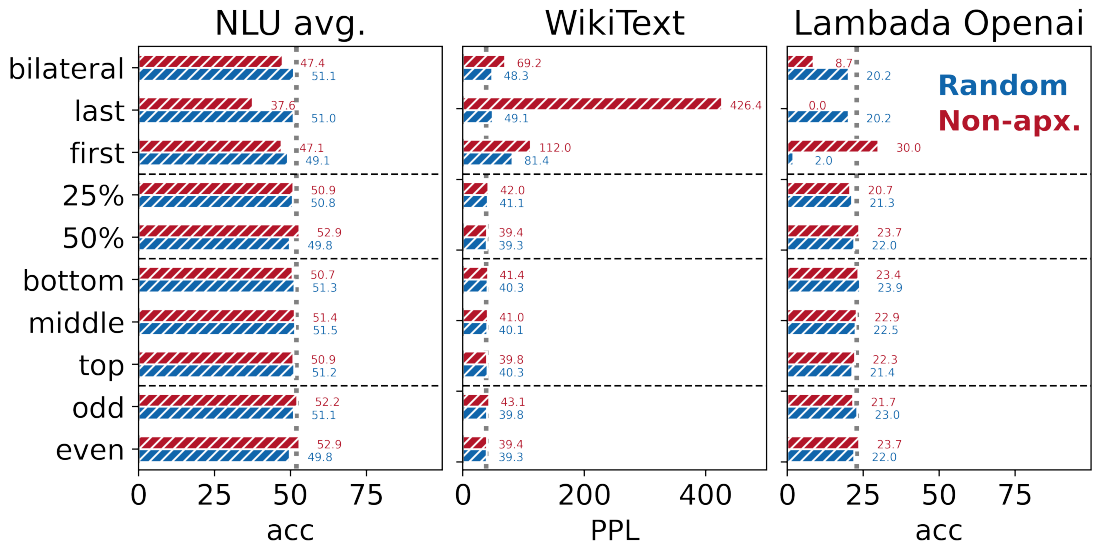


Figure 4.3: Performance of *RndEmbQK* and *Non-approximate* across nine hybrid configurations. The vertical dotted lines represent the *Standard* baseline.

using *Standard* attention mechanism only in the last layer greatly harms performance, leading to PPL exceeding 400 on WIKITEXT. This indicates that the normalisation strength provided by a single *Standard* layer is limited. Therefore, in extreme hybrid settings where we can afford only one or two *Standard* layers, we should choose a substitute that still respects the main design principles presented in the *uniform* setting (i.e. a stronger lightweight attention). Conversely, if the compute budget allows using even a small fraction of *Standard* transformer layers (e.g. 25%), we can safely replace the remainder with a much simpler mechanism and still maintain competitive accuracy.

4.7 Conclusion

We systematically relax core design principles in a controlled setting, offering the first principled framework for assessing which aspects of attention are truly foundational and which can be safely simplified in language modeling. Our findings reveal that adhering to standard attention design principles varies between *uniform* and *hybrid* architectures. Token mixing and following the mathematical form are crucial for attention alternatives when applied uniformly, but not necessary for *hybrid*. Strategically integrating a few standard attention layers within LMs can greatly improve, even overcome, the limitations of less powerful attention mechanisms. This is likely due to the inherent normalisation of standard attention, fostering training stability.

Limitations

We performed experiments using a maximum model size of 500M parameters and a pretraining budget of 15B tokens, using a monolingual tokeniser and vocabulary, similar to Allal et al. (2025); Poli et al. (2023). While experimenting with larger models and different model families presents interesting avenues for future work, we believe that the current scope sufficiently supports our conclusions regarding the relative effectiveness of different attention designs.

Acknowledgments

This projected made use of time on UK Tier-2 HPC facility JADE@ARC, funded by EPSRC (EP/T022205/1). We would like to thank Miles Williams and Atsuki Yamaguchi for their invaluable feedback.

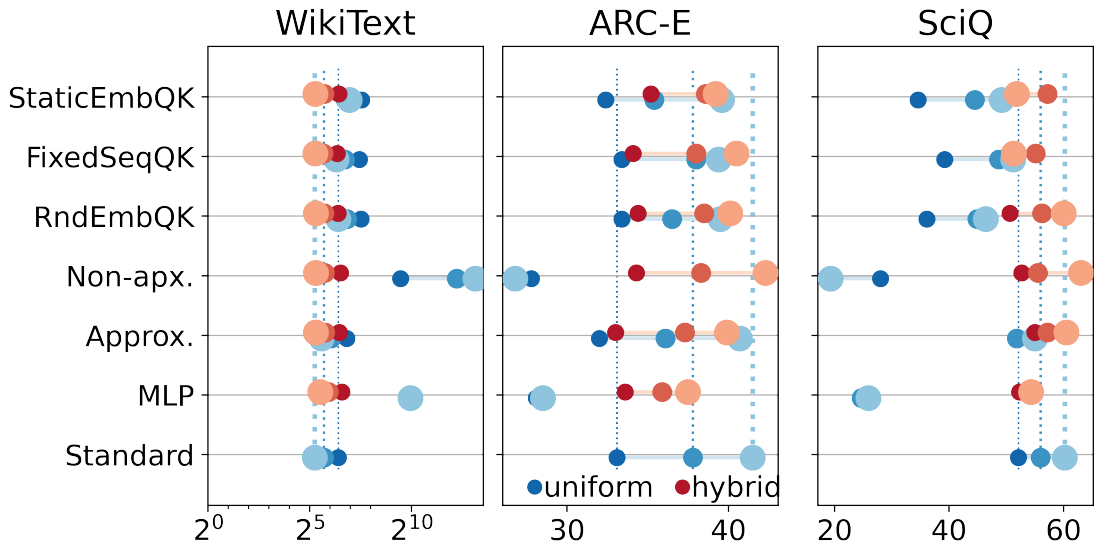


Figure 4.4: Predictive performance of 70M parameters (small dots), 160M parameters (medium dots), and 500M parameters (large dots) models with different attention mechanisms and configurations on WIKITEXT, ARC-E, and SCIQ.

4.8 Appendix A: Experiments with Different Model Sizes

To assess the impact of model size, we evaluate all attention mechanisms across models with approximately 70M, 160M, and 500M parameters. Figure 4.4 illustrates the predictive performance of these models on the WIKITEXT, ARC-E, and SCIQ datasets. Our results indicate that the predictive performance of LMs with a *hybrid* configuration consistently improves with increasing model size. For instance, the accuracy of the *Non-approximate* method on ARC-E improves from 34.3 to 42.3 when increasing the model size from 70M to 500M. Furthermore, all attention mechanisms incorporating token mixing achieve predictive performance comparable to a same-sized model employing *standard* attention (indicated by the vertical dotted lines in Figure 4.4). For *RndEmbQK*, such performance gap on WIKITEXT PPL is even within 1.2 across all sizes. This trend suggests that our observations may generalise to larger models.

To further investigate the immediate generalizability of our findings, we further pretrain a larger model (Yang et al., 2025a, Qwen3-1.7b-Base) from scratch on 45 billion tokens with *Standard* and our proposed *RndEmbQK* and *Non-approximate* variants in both uniform and hybrid configurations. Table 4.2 presents their performance on NLU and LM tasks. We find both *RndEmbQK* and *Non-apx.* under *hybrid* configuration, achieve performance comparable to *Standard* across all downstream tasks, which is consistent to our observation on models with modest scales. However, different to the model with 500M parameters, *Non-approximate* under *uniform* configuration successfully converges. This is because Qwen3 incorporates RSMNorm above the queries and key in its attention module. This normalisation helps to alleviate the

		ARC-E	BoolQ	COPA	PiQA	SciQ	RTE	HellaSwag	Avg.	Wiki	LAMBADA	
		acc↑	acc↑	acc↑	acc↑	acc↑	acc↑	acc↑	acc↑	ppl↓	ppl↓	acc↑
	Rnd. Guess	$25.0_{0.0}$	$50.0_{0.0}$	$50.0_{0.0}$	$50.0_{0.0}$	$25.0_{0.0}$	$50.0_{0.0}$	$25.0_{0.0}$	39.9	3E+5	3E+6	$0.0_{0.0}$
	Majority	$25.7_{0.0}$	$62.2_{0.0}$	$56.0_{0.0}$	$50.5_{0.0}$	$25.0_{0.0}$	$52.7_{0.0}$	$25.0_{0.0}$	39.9	-	-	-
	Standard	$44.6_{1.0}$	$56.4_{0.9}$	$64.0_{4.8}$	$64.0_{1.1}$	$67.3_{1.5}$	$52.7_{3.0}$	$30.5_{0.5}$	54.2	27.6	60.0	$28.9_{0.6}$
UNI.	Non-apx.	$41.0_{1.0}$	$61.9_{0.9}$	$58.0_{5.0}$	$59.5_{1.2}$	$56.9_{1.6}$	$52.4_{3.0}$	$27.8_{0.5}$	51.1	67.3	619.6	$8.1_{0.4}$
UNI.	RndEmbQK	$44.4_{1.0}$	$50.2_{0.9}$	$60.0_{4.9}$	$62.4_{1.1}$	$56.3_{1.6}$	$54.9_{3.0}$	$28.6_{0.5}$	51.0	54.9	1872.8	$3.8_{0.3}$
HYB.	Non-apx.	$\mathbf{45.0}_{1.0}$	$58.1_{0.9}$	$65.0_{1.8}$	$63.4_{1.1}$	$\mathbf{66.2}_{1.5}$	$53.1_{3.0}$	$\mathbf{30.2}_{0.5}$	54.4	29.9	77.4	$\mathbf{26.8}_{0.6}$
HYB.	RndEmbQK	$45.4_{1.0}$	$57.0_{0.9}$	$67.0_{4.7}$	$64.5_{1.1}$	$65.5_{1.5}$	$55.2_{3.0}$	$30.4_{0.5}$	55.0	28.0	61.6	$29.8_{0.6}$

Table 4.2: Performance of *uniform*, *hybrid* and *standard* models (1.7B). Blue (Non-apx.) and green (RndEmbQK) denote variants that relax `Mathematical Form`, and `Sequence-Dependency`, respectively. ↑ and ↓ denote that higher and lower values are preferred, respectively.

potential for pre-softmax attention activations to explode, but it is less effective than using several standard layers, as it restricts the length of query and key vectors, narrowing the adaptable range for raw pre-softmax activations.

4.9 Appendix B: Grouped-query Attention Ablation

To confirm the generality of our main investigations, we also trained 500M parameter versions of the *Standard*, *Non-approximate*, and *RndEmbQK* models using the grouped-query attention configuration. These models are trained on the same 15 billion tokens, with precisely matched parameter counts. We observe that the results on downstream tasks remain consistent across both the multi-head attention and grouped-query attention configurations. Their performance on both NLU and LM tasks is detailed in Table 4.3.

4.10 Appendix C: Robustness to Context Length

Table 4.4 illustrates the perplexity scores of the UNIFORM, HYBRID and *standard* models on WIKITEXT dataset. These models were evaluated across various contextual lengths (128, 256, 512, 1024, and 2048 tokens), all while being trained on a maximum sequence length of 2048 tokens. The results clearly show that models incorporating token mixing achieve lower perplexity scores with longer contexts. This indicates their ability to capture more contextual information for predicting the next token. Furthermore, under the *hybrid* configuration, the perplexity scores for the *RndEmbQK*, *FixedSeqQK*, *StaticEmbQK*, *Approximate* and *Non-approximate* attention mechanisms consistently match those of the *standard* model on WIKITEXT, regardless of contextual length.

		ARC-E	BoolQ	COPA	PiQA	SciQ	RTE	HellaSwag	Avg.	Wiki	LAMBADA	
		acc↑	acc↑	acc↑	acc↑	acc↑	acc↑	acc↑	acc↑	ppl↓	ppl↓	acc↑
	Rnd. Guess	$25.0_{0.0}$	$50.0_{0.0}$	$50.0_{0.0}$	$50.0_{0.0}$	$25.0_{0.0}$	$50.0_{0.0}$	$25.0_{0.0}$	39.9	3E+5	3E+6	$0.0_{0.0}$
	Majority	$25.7_{0.0}$	$62.2_{0.0}$	$56.0_{0.0}$	$50.5_{0.0}$	$25.0_{0.0}$	$52.7_{0.0}$	$25.0_{0.0}$	39.9	-	-	-
	Standard	$39.4_{1.0}$	$49.6_{0.9}$	$60.0_{5.0}$	$62.2_{1.1}$	$59.3_{1.6}$	$51.6_{3.0}$	$28.1_{0.5}$	50.0	38.6	154.0	$22.9_{0.6}$
UNI.	Non-apx.	$26.8_{0.9}$	$37.8_{0.8}$	$52.0_{5.0}$	$52.0_{1.2}$	$20.3_{1.3}$	$52.7_{3.0}$	$25.9_{0.4}$	38.2	5466.8	2E+6	$0.0_{0.0}$
	RndEmbQK	$37.9_{1.0}$	$53.2_{0.9}$	$56.0_{5.0}$	$58.3_{1.2}$	$46.7_{1.6}$	$52.7_{3.0}$	$27.2_{0.4}$	47.4	84.6	6462.7	$12.4_{0.2}$
HYB.	Non-apx.	$40.7_{1.0}$	$44.6_{0.9}$	$67.0_{5.0}$	$61.3_{1.1}$	$61.5_{1.5}$	$52.4_{3.0}$	$28.3_{0.5}$	50.8	38.1	133.1	$23.4_{0.6}$
	RndEmbQK	$40.1_{1.0}$	$45.8_{0.9}$	$63.0_{4.9}$	$61.3_{1.1}$	$61.8_{1.5}$	$52.7_{3.0}$	$28.3_{0.5}$	50.4	39.3	138.6	$23.6_{0.6}$

Table 4.3: Performance of *uniform*, *hybrid* and *standard* models (500m) using grouped-query attention. Blue (Non-apx.) and green (RndEmbQK) denote variants that relax `Mathematical Form`, and `Sequence-Dependency`, respectively. ↑ and ↓ denote that higher and lower values are preferred, respectively.

PPL↓	length	128	256	512	1024	2048
	Standard	69.9	56.2	47.8	42.0	38.1
UNIFORM	MLP	993.5	993.5	993.5	993.5	993.5
	Approx.	81.7	66.4	57.2	51.2	47.9
	Non-apx.	10023.7	9476.8	9173.5	9064.8	9025.9
	RndEmbQK	107.1	95.7	89.6	86.3	84.8
	FixedSeqQK	100.5	89.6	83.6	80.6	79.1
	StaticEmbQK	104.8	92.2	85.5	81.7	79.9
HYBRID	MLP	81.3	66.4	57.0	50.3	45.8
	Approx.	71.8	57.8	49.3	43.4	39.4
	Non-apx.	69.0	56.0	48.0	42.5	39.4
	RndEmbQK	72.4	58.1	49.4	43.4	39.3
	FixedSeqQK	69.0	56.0	48.0	42.3	38.5
	StaticEmbQK	70.1	56.6	48.4	42.6	38.7

Table 4.4: Perplexities of *uniform*, *hybrid* and *standard* models (500M) on WIKITEXT across different context lengths. ↑ and ↓ denote that higher and lower values are preferred, respectively.

4.11 Appendix D: Characteristics of Different Simpler Attentions

Unlike previous work that primarily focused on reducing computational time complexity to sub-quadratic with respect to contextual sequence length, we define "simpler attention" more broadly. This encompasses mechanisms that reduce time complexity concerning any factor: inference batch size, sequence length, or hidden dimension. Below, we systematically summarise the characteristics of the different simpler attention mechanisms we investigated.

RndEmbQK and FixedSeqQK. These mechanisms create global static attention graphs during inference. This approach reduces the computational time complexity and cache size within attention while enabling batched decoding (see section 4.12 and 4.15).

StaticEmbQK. Inspired by cross-layer attention sharing (Rajabzadeh et al., 2024; Xiao et al., 2019), this mechanism primarily captures semantic similarities between input tokens without contextualization. It establishes an upper bound for broadcasting attention matrices from initial layers to all subsequent layers by aligning its parameter count with standard attention. While *StaticEmbQK* attention does not explicitly reduce computational time complexity, it allows for system optimisation by computing attention scores asynchronously. This enables scores to be prefetched before sequentially retrieving output hidden states from each layer.

Approximate and Non-approximate. These attention mechanisms result in time complexities linear to sequence length. Their recurrent forms are detailed in section 4.16. *Non-approximate* can further reduce the activation memory, cache size, and floating-point operations per iteration (FLOPs/it) required for large LMs during the decode stage, offering advantages over *Approximate*. The details for these reductions are provided in Appendices 4.14, 4.15, and 4.13, respectively.

4.12 Appendix E: Time Complexities in Attention Computation

Table 4.5 details the computational time complexity for a single forward pass, explicitly excluding any caching mechanisms. For *RndEmbQK* and *FixedSeqQK* attention, which employ global attention scores, the floating-operations could be further reduced to through pre-computation and subsequent caching of these scores (see section 4.13). This optimisation would free up computational resources, enabling further software-level enhancements such as coordinating CPUs and GPUs to pre-fetch the pre-calculated attention scores. While *StaticEmbQK* does not inherently offer a lower computational time complexity, it provides an upper bound for pre-computing attention scores on static embeddings by aligning the number of parameters. If attention scores on static embeddings are pre-computed, the computational time complexity would be reduced by $O\left((l-1)\cdot(BL^2d+BLd^2)\right)$ in total, where l represents the total number of Transformer layers. Furthermore, an attention mechanism that supports pre-computation offers the potential to proactively evict values, which could lead to further reductions in computation, particularly if the attention matrices exhibit sparsity.

Attention	Complexity $\mathcal{O}(.)\downarrow$
Standard	$BL^2d + BLd^2$
MLP	BLd^2
Approx.	BLd^2
Non-apx.	BLd^2
RndEmbQK	$BL^2d + BLd^2$
FixedSeqQK	$BL^2d + BLd^2$
StaticEmbQK	$BL^2d + BLd^2$

Table 4.5: Details of time complexities for each attention across all attention variants, where h denotes the number of attention heads, B denotes the batch size, L denotes the input sequence length, d denotes the hidden dimension. We assume $d = h \times d_h$, where h is the number of attention heads and d_h is the dimension of each attention head. We also ignore those low-order terms for element-wise activations and scaling factors with a $\mathcal{O}(BLd)$ complexity. $\uparrow$ and $\downarrow$ denote that higher and lower values are preferred, respectively.

4.13 Appendix F: Floating-point Operations per Token

Table 4.6 details the floating-point operations per iteration (FLOP/it) for inference with the cache enabled. We focus solely on General Matrix Multiplications (GEMMs) (Narayanan et al., 2021, GEMMs), as they are the dominant contributors to the total floating-point operations.

Non-approximate achieves a low FLOP/it, equivalent to that of the simplest *MLP* model, because it leverages vectors instead of the matrices employed by the Approximate method for state tracking. This structural difference significantly reduces the number of GEMMs required.

Furthermore, if *RndEmbQK* and *FixedSeqQK* are allowed to use pre-computed global attention scores, their FLOP/it can be further reduced. During the prefill stage, the operations drop to $2L^2d + 2BLd^2$ and $2BLd + 2d^2$ during prefill and decode stage respectively.

4.14 Appendix G: Activation Memory Required for Attention Computation

We detail the activation memory required for half-precision training in Table 4.7. Unlike the full recomputation method mentioned in Smith et al. (2022), our approach incorporates sequence parallelism following Korthikanti et al. (2023). We find that *RndEmbQk* and *FixedSqeQK* are effective at reducing activation memory, particularly when using a substantially large batch size. Furthermore, both *Approximate* and *Non-approximate* enhance memory efficiency for

Attention	Prefill↓	Decode↓
Standard	$4BL^2d + 6BLd^2$	$6Bd^2 + 4BLd$
MLP	$6BLd^2$	$6Bd^2$
Approx.	$14BLd^2$	$10Bd^2$
Non-apx.	$6BLd^2$	$6Bd^2$
RndEmbQK	$2L^2d + 2BL^2d + 6BLd^2$	$2Ld + 2BLd + 6Bd^2$
FixedSeqQK	$2L^2d + 2BL^2d + 6BLd^2$	$2Ld + 2BLd + 6Bd^2$
StaticEmbQK	$4BL^2d + 6BLd^2$	$6Bd^2 + 4BLd$

Table 4.6: Details of floating-point operations per iteration for each attention across all attention variants, where h denotes the number of attention heads, B denotes the batch size, L denotes the input sequence length, d denotes the hidden dimension. We assume $d = h \times d_h$, where h is the number of attention heads and d_h is the dimension of each attention head. ↑ and ↓ denote that higher and lower values are preferred, respectively.

Attention	Activation memory↓
Standard	$(8BLd + 2BL^2h)/t$
MLP	$8BLd/t$
Approx.	$11BLd/t + 3Bd^2/ht$
Non-apx.	$(8BLd + 4BLh)/t$
RndEmbQK	$(4BLd + 8Ld + 2L^2h)/t$
FixedSeqQK	$(4BLd + 8Ld + 2L^2h/)t$
StaticEmbQK	$(8BLd + 2BL^2h)/t$

Table 4.7: Details of activation memory for each attention across all attention variants, where h denotes the number of attention heads, B denotes the batch size, L denotes the input sequence length, d denotes the hidden dimension, t denotes the tensor parallel size. We assume $d = h \times d_h$, where h is the number of attention heads and d_h is the dimension of each attention head. We ignore the attention dropout here. ↑ and ↓ denote that higher and lower values are preferred, respectively.

long-context processing. *Non-approximate* offers a superior reduction in activation memory compared to *Approximate*, especially for large LMs characterised by a relatively large hidden state dimension.

Attention	Cache Size for Inference↓
Standard	$4BLd$
MLP	0
Approx.	$6Bd + 4Bd^2/h$
Non-apx.	$2Bd + 4Bh$
RndEmbQK	$2(B+1)Ld$
FixedSeqQK	$2(B+1)Ld$
StaticEmbQK	$4BLd$

Table 4.8: Details of cache size (in bytes) per layer across all attention variants required during inference, where h denotes the number of attention heads, B denotes the batch size, L denotes the context length, d denotes the hidden dimension. We assume $d = h \times d_h$, where h is the number of attention heads and d_h is the dimension of each attention head. $\uparrow$ and $\downarrow$ denote that higher and lower values are preferred, respectively.

4.15 Appendix H: Cache Size Required for Inference

Table 4.8 presents the cache size required for half-precision inference. Both the *Approximate* and *Non-approximate* variants allow the cache size to be independent of the context sequence length. Meanwhile, *RndEmbQk* and *FixedSeqQK* can reduce the cache size by nearly half by sharing the same set of keys within the same batch, provided the batch size is sufficiently large. It is also important to note that *RndEmbQK* and *FixedSeqQk* enable a cache size further optimised to $(2L + \delta)\delta$. This can be achieved by using a dynamic cache and prefetching the attention scores for the next δ steps into a buffer, given that the attention matrices are independent of the inputs.

4.16 Appendix I: Recurrent Form of Linear Attentions

The recurrent form of the *Approximate* attention computation, derived from Eq. 4.6, is presented in Eq. 4.17. Similarly, Eq. 4.18 shows the recurrent form of the *Non-approximate* attention computation, originating from Eq. 4.7. As detailed in Table 4.5, the *Approximate* attention mechanism necessitates the computation of recursions for both first-order and second-order terms in the Taylor expansion, resulting in a higher time complexity compared to the *Non-approximate* approach. A key characteristic of $\mathbf{O}_i$ in Eq. 4.18 is that its denominator strictly increases with the index i. Notably, as i grows along the sequence, the attention score for the i^{th} token, given by $\frac{e^{q_i k_i^\top} v_i}{\sum_{j=1}^{i-1} e^{q_j k_j^\top} + e^{q_i k_i^\top}}$, becomes progressively more challenging to increase.

Hyperparameters in Pretraining	
Maximum train steps	120000
Batch size (in total)	256 instances
Adam $epsilon$	1e-8
Adam β_1	0.9
Adam β_2	0.9999
Sequence length	2048
Peak learning rate	4e-4 (3e-4 for Qwen3-1.7B)
Learning rate schedule	CosineLRScheduler
Number of cycles in scheduler	0.5
Warmup steps	2000 (1B tokens)
Weight decay	0.1
Max gradient norm clip value	1.0

Table 4.9: Details of hyperparameters used in pre-training.

$$\mathbf{o}_i = \mathbf{o}_{0i} + \mathbf{o}_{1i} + \mathbf{o}_{2i} \tag{4.14}$$

$$\mathbf{o}_{0i} = \frac{\sum_{j=1}^{i-1} v_j + v_i}{i} \tag{4.15}$$

$$\mathbf{o}_{1i} = \frac{q_i \left(\sum_{j=1}^{i-1} k_j^\top v_j + k_i^\top v_i \right)}{q_i \left(\sum_{j=1}^{i-1} k_j^\top + k_i^\top \right)} \tag{4.16}$$

$$\mathbf{o}_{2i} = \frac{\frac{q_i^2}{\sqrt{2}} \left(\sum_{j=1}^{i-1} (\frac{k_j^2}{\sqrt{2}})^\top v_j + (\frac{k_i^2}{\sqrt{2}})^\top v_i \right)}{\frac{q_i^2}{\sqrt{2}} \left(\sum_{j=1}^{i-1} (\frac{k_j^2}{\sqrt{2}})^\top + (\frac{k_i^2}{\sqrt{2}})^\top \right)} \tag{4.17}$$

$$\mathbf{o}_i = \frac{\sum_{j=1}^{i-1} \mathrm{e}^{q_j k_j^\top} v_j + \mathrm{e}^{q_i k_i^\top} v_i}{\sum_{j=1}^{i-1} \mathrm{e}^{q_j k_j^\top} + \mathrm{e}^{q_i k_i^\top}} \tag{4.18}$$

4.17 Appendix J: Hyperparameters

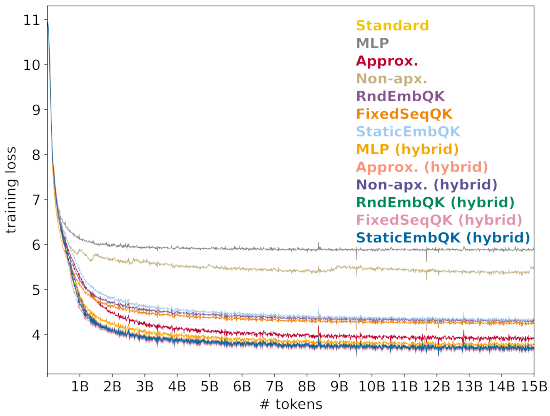
The hyperparameters used in pre-training are listed in Table 4.9.

4.18 Appendix K: Training Loss across all Attention Mechanisms

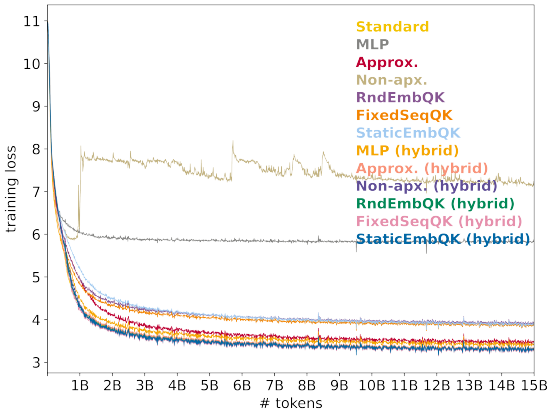
Figure 4.5 presents the loss curves across all model variants and sizes, while training for 15B tokens.

4.19 Appendix L: Model Configurations for Different Sizes

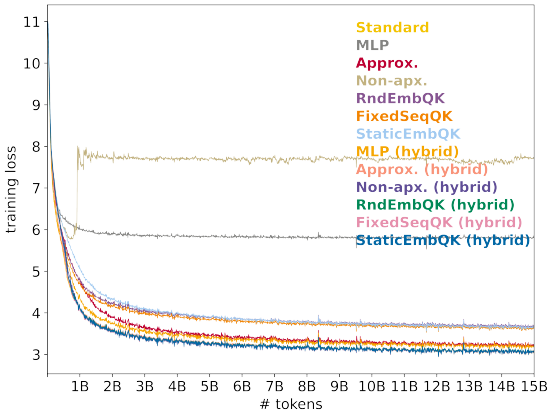
Table 4.10 presents the detailed configurations of models across various sizes (70M, 160M and 500M and 1.7B).



(a) Training loss across models with 70M parameters



(b) Training loss across models with 160M parameters



(c) Training loss across models with 500M parameters

Figure 4.5: Training loss across all model variants with three different sizes.

Model Size	70M	160M	500M	1.7B
Hidden Size	512	768	896	2048
Intermediate Size	2048	3072	4864	6144
Num of Hidden Layers	6	12	24	28
Max Window Layers	6	12	24	28
Num of Attention Heads	8	12	14	16
Num of Key Value Heads	8	12	14	16

Table 4.10: Details of model configurations for different sizes.

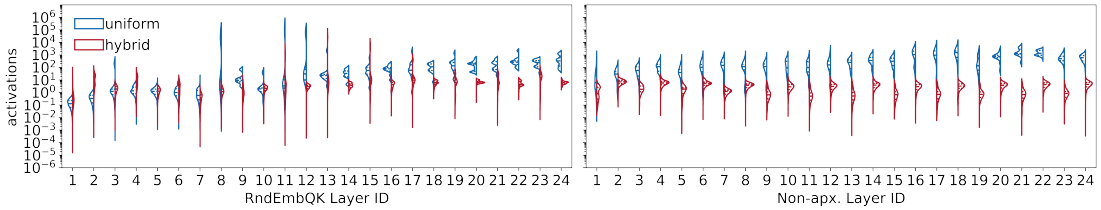


Figure 4.6: Distribution of raw logits in the pre-softmax activations for *RndEmbQK* (left) and *Non-approximate* (right) attention mechanisms in both uniform and hybrid configurations.

4.20 Appendix M: Distribution of Raw Logits

Figure 4.6 (the full version of Figure 4.2) exhibits the magnitude of pre-softmax activations within each 24-layer (500M) *RndEmbQK* and *Non-approximate* layer in the *hybrid* configuration.

4.21 Appendix N: Attention Characteristics from All Layers across Attention Variants

Figure 4.7, the full version of the left subfigure in Figure 4.1), exhibits attention characteristics from all 24 layers across *Standard* attention and five attention variants - *RndEmbQK*, *FixedSeqQK*, *StaticEmbQK*, *Approximate* and *Non-approximate*.

4.22 Appendix O: Attention Characteristics from All Layers across Configurations

Figure 4.8, the full version of the right subfigure in Figure 4.1, exhibits attention characteristics from all 24 layers across *Standard* and two representative attention variants - *Approximate* and *Non-approximate* in both *uniform* and *hybrid* configurations.

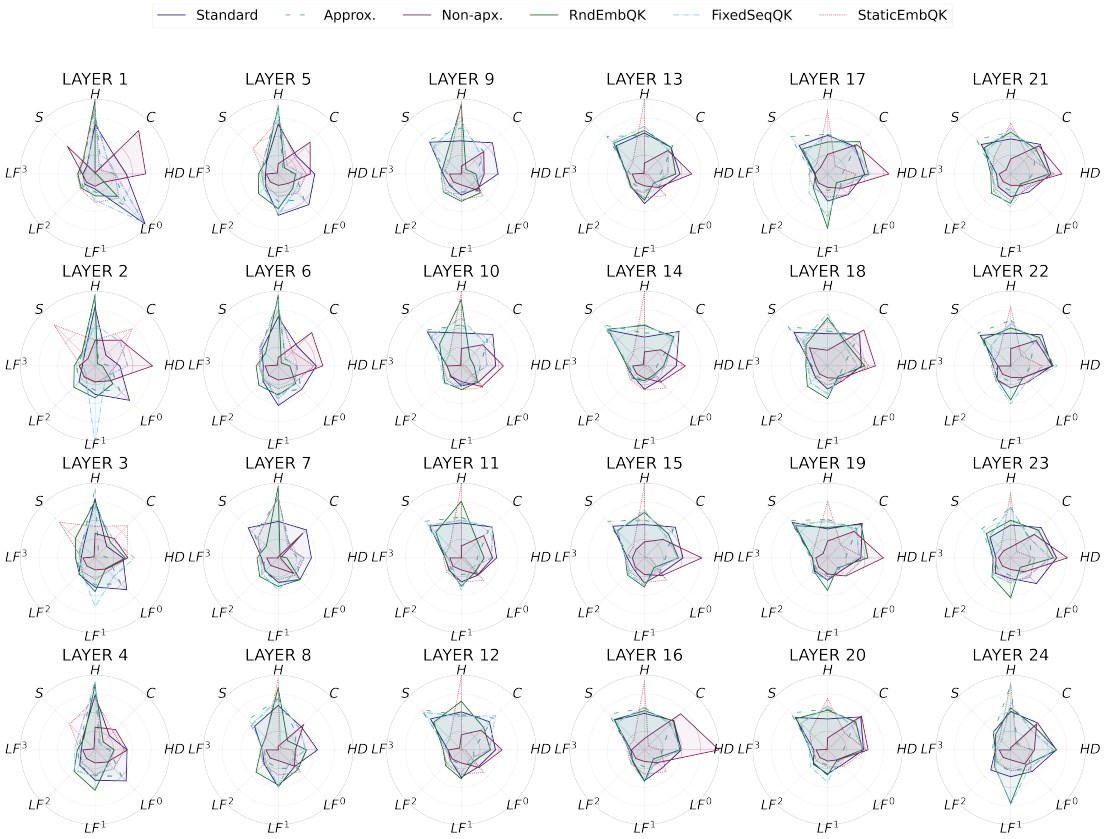


Figure 4.7: Visualization of attention matrix characteristics across different layers for *Approximate*, *Non-approximate*, *RndEmbQK*, *FixedSeqQK* and *StaticEmbQK*, and their hybrid variants, compared to *Standard* (H: ENTROPY, C: CONC, HD: HEADDIV, LF: LOCFOCN, S: SINK).

4.23 Appendix P: Model Configurations for Ablation Study

Table 4.11 details nine distinct *hybrid* architectures, as discussed in § 4.6, for 24-layer model variants with approximately 500 million parameters.

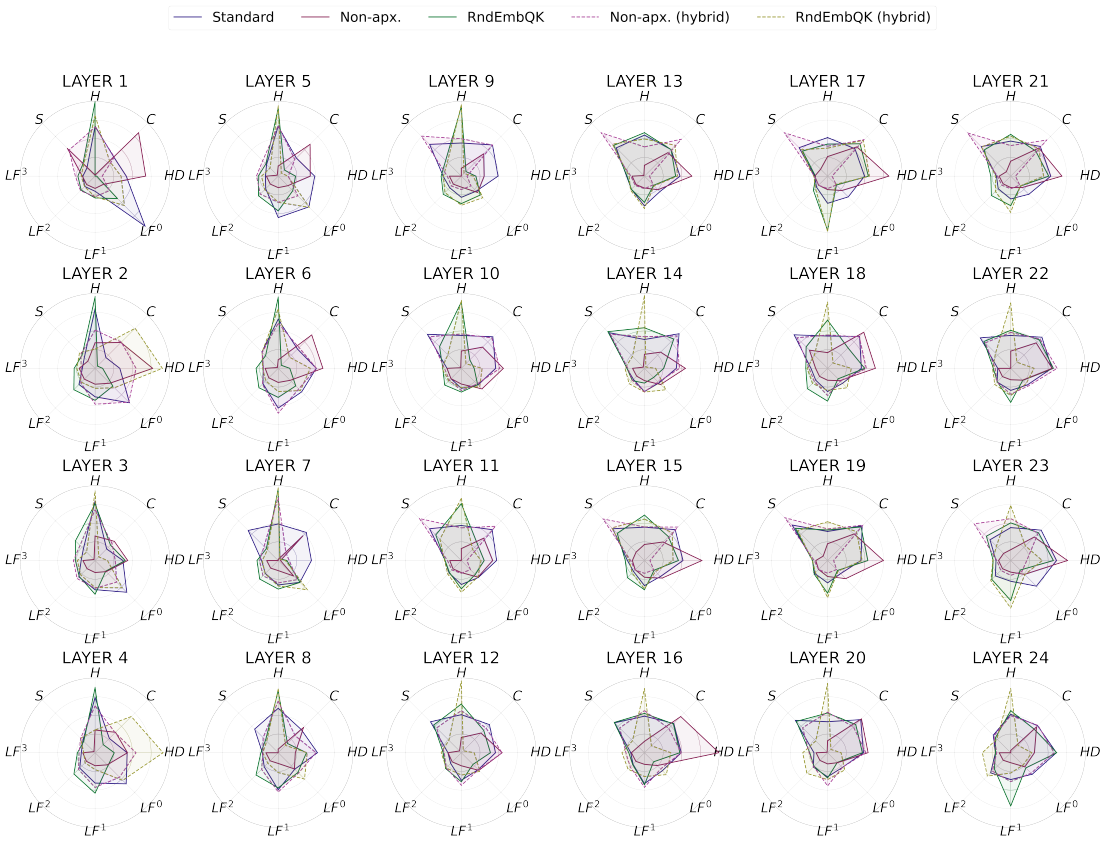


Figure 4.8: Visualization of attention matrix characteristics across different layers for *Non-approximate* and *RndEmbQK*, and their hybrid variants, compared to *Standard* (H: ENTROPY, C: CONC, HD: HEADDIV, LF: LOCFOCN, S: SINK).

Config	Standard Layer IDs
even (50%)	{2,4,6,8,10,12,14,16,18,20,22,24}
odd	{1,3,5,7,9,11,13,15,17,19,21,23}
top	{1,2,3,4,5,6,7,8,9,10,11,12}
middle	{1,2,3,4,5,6,19,20,21,22,23,24}
bottom	{13,14,15,16,17,18,19,20,21,22,23,24}
25%	{4,8,12,16,20,24}
first	{1}
last	{24}
bilteral	{1,24}

Table 4.11: Details of model configurations for ablation study.

Chapter 5

Conclusion

This thesis explored efficient methods for Transformer-based language models by optimising their classic components to test the reducibility of each. Through three empirical studies, each targeting a specific component, we developed novel approaches that will guide the design and optimisation of future language model architectures.

In Chapter 2, we introduced HASHFORMERS, a new family of pre-trained transformers that are vocabulary-independent. This allows them to handle an unlimited vocabulary (i.e. all possible tokens in a corpus) using a much smaller, fixed-size embedding matrix. In addition to proposing computationally cheap hashing functions that group individual tokens into embeddings, we developed three variants that eliminate the need for an embedding matrix entirely, which further reduces memory requirements. Our empirical results showed that HASHFORMERS are more memory-efficient than standard pre-trained transformers, all while maintaining comparable predictive performance when fine-tuned on various text classification tasks. Our findings later inspired work by Deiseroth et al. (2024) on creating subword tokeniser-free generative language models.

Chapter 3 shifted to explore a method for reducing parameter redundancy across multiple subspaces for generating multiple attention heads. Inspired by the superposition principle in Quantum Mechanics, we analogised both the token representations at different positions and the hidden states from different attention heads as quantum superpositions. This analogy led us to hypothesise that the multi-head attention (MHA) mechanism in a transformer-based language model could be improved by using a more parameter-efficient method. Instead of head-wise projections, we proposed a novel approach that replaces them with a set of combinations between a single "seed state" and different head-wise states, allowing for a more efficient MHA mechanism. Our proposed method is substantially more memory-efficient than other MHA alternatives while maintaining high predictive performance on various downstream tasks. It

requires only a negligible number of additional parameters compared to the standard MHA. For context, IA^3 (Liu et al., 2022), an earlier work, also used head-wise embeddings, but for incremental parameter-efficient fine-tuning. Similarly, GQA (Ainslie et al., 2023a), a concurrent and highly cited work with a similar goal, grouped several different query heads to share a single key-value pair, but this approach requires a larger parameter count.

Finally, Chapter 4 delves into the form of the token mixer within the attention mechanism of transformer-based language models. We deconstructed the standard dot-product attention, abstracting four key design principles: mixing information across positions, sequence-dependency, a specific mathematical form, and the coupling of queries and keys to evolving hidden states. To test the necessity of each principle, we systematically relaxed them, applying these changes both uniformly across all layers and in hybrid architectures. Our empirical results show that mechanisms for mixing tokens are indispensable; their absence causes models to perform near-randomly. However, the exact mathematical form and sequence dependency can be significantly relaxed, particularly when a subset of layers retains the standard attention mechanism. Interestingly, we also found that variants which fail on their own can achieve robust performance when interleaved with standard attention, highlighting a cooperative effect. We anticipate that our findings will deepen the understanding of what truly underpins attention's effectiveness and pave the way for simplifying language models without sacrificing performance in the future.

From these three studies, several overarching insights emerge:

- Embedding layers in encoder-only language models can be greatly compressed with only minor performance degradation on downstream tasks. Projection-based embedding compression using techniques like multihashing is generally more effective than simply sharing entire embedding vectors across different tokens. The model's capabilities could be further enhanced by incorporating better feature extraction for the hashing process. Decoupling the pre-training objective from a vocabulary-dependent token ID prediction is a crucial step that enables this training approach.

- The commonalities among different model components allow developers to apply lessons learned from one design to optimise others, taking our multiple head embeddings (Xue and Aletras, 2023) as an example. This work also indicates that using different linear projections to generate multiple subspaces is an inefficient approach. This suggests that there is an opportunity for optimisation by leveraging element-wise operations and parameter sharing instead.

- A "deconstructionist" approach offers new guidance for model design by blurring the traditional boundaries between model types (e.g., attention-based, graph-based, or recurrent). By abstracting deeper, more fundamental attributes from a component's surface

form and systematically investigating them through controlled experiments, we can better understand its core design principles and guide future optimisations.

This thesis introduces several promising areas for future research:

- **New pre-training objectives for decoder-only models.** Pre-training for decoder-only models, which typically involves next-token prediction, is not compatible with non-bijective hashing functions. This is because these functions prevent the model's outputs from being accurately mapped back to their original tokens. Therefore, a new pre-training objective should be designed to accommodate hash-indexed or encrypted tokens, allowing for more efficient language modeling.

- **Re-evaluating the need for tokenisation.** The hashing process used in HASHFORMERS can be viewed as an unlearned form of multi-bit quantisation, with the learnable projection in the embedding layer acting as de-quantisation. If unlearned hashing is replaced with a shallow neural network for feature extraction, tokenisation might become a redundant step. This suggests that tokenisation could be more efficiently integrated directly into the neural network architecture.

- **Leveraging design principles across network components.** Our research demonstrates the value of applying design principles from one network component to another, as seen in our use of absolute position embeddings to optimise the multi-head attention mechanism. There is significant potential to further explore this approach. Recent advancements in attention mechanisms, layer normalisation, and feed-forward layers offer valuable insights that might be transferred to optimise other parts of the network architecture.

- **Exploring Hybrid Attention Architectures.** Our findings suggest that a hybrid approach combining different attention variants can maintain predictive performance. Keeping a few standard attention layers within the model is crucial for this. Given that input lengths vary, a promising future direction is to design efficient language models that integrate attention variants optimised for specific characteristics, such as context length, batch size, hidden dimensions, and cross-device coordination.

- **Building More Efficient Multilingual Language Models.** Our findings highlight promising optimisation strategies for multilingual training and inference. To address the over-segmentation issues common in models with constrained embedding memory, adopting a coarser-grained tokeniser (such as a full-word tokeniser) can improve both memory efficiency and inference speed by reducing sequence lengths for low-resource languages. Alternatively, efficiency can be improved through hybrid architectures. By integrating memory-efficient linear attention mechanisms, models can more effectively process (i.e. encode and decode) the long sequences generated by fine-grained (e.g., byte-level)

tokenisers.

Bibliography

Aghajanyan, A., Zettlemoyer, L., and Gupta, S. (2020). Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*.

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebron, F., and Sanghai, S. (2023a). GQA: Training generalized multi-query transformer models from multi-head checkpoints. In Bouamor, H., Pino, J., and Bali, K., editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901, Singapore. Association for Computational Linguistics.

Ainslie, J., Lei, T., de Jong, M., Ontañón, S., Brahma, S., Zemlyanskiy, Y., Uthus, D., Guo, M., Lee-Thorp, J., Tay, Y., Sung, Y.-H., and Sanghai, S. (2023b). Colt5: Faster long-range transformers with conditional computation.

Aksenov, Y., Balagansky, N., Lo Cicero Vaina, S., Shaposhnikov, B., Gorbatovski, A., and Gavrilov, D. (2024). Linear transformers with learnable kernel functions are better in-context models. In Ku, L.-W., Martins, A., and Srikumar, V., editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9584–9597, Bangkok, Thailand. Association for Computational Linguistics.

Alajrami, A. and Aletras, N. (2022). How does the pre-training objective affect what large language models learn about linguistic properties? In Muresan, S., Nakov, P., and Villavicencio, A., editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 131–147, Dublin, Ireland. Association for Computational Linguistics.

Allal, L. B., Lozhkov, A., Bakouch, E., Blázquez, G. M., Penedo, G., Tunstall, L., Marafioti, A., Kydlícek, H., Lajarín, A. P., Srivastav, V., et al. (2025). Smollm2: When smol goes big-data-centric training of a small language model. *CoRR*.

Ankner, Z., Parthasarathy, R., Nrusimha, A., Rinard, C., Ragan-Kelley, J., and Brandon, W. (2024). Hydra: Sequentially-dependent draft heads for medusa decoding. In *First Conference on Language Modeling*.

Arora, S., Eyuboglu, S., Zhang, M., Timalsina, A., Alberti, S., Zou, J., Rudra, A., and Re, C. (2024). Simple linear attention language models balance the recall-throughput tradeoff. In Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F., editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 1763–1840. PMLR.

Bae, S., Ko, J., Song, H., and Yun, S.-Y. (2023). Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding. In Bouamor, H., Pino, J., and Bali, K., editors, *Proceedings of the 2023 Conference on Empirical Methods in*

Natural Language Processing, pages 5910–5924, Singapore. Association for Computational Linguistics.

Bai, H., Hou, L., Shang, L., Jiang, X., King, I., and Lyu, M. R. (2022). Towards efficient post-training quantization of pre-trained language models. *Advances in Neural Information Processing Systems*, 35:1405–1418.

Baker, A. (2022). Simplicity. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2022 edition.

Behrouz, A., Zhong, P., and Mirrokni, V. (2024). Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*.

Beltagy, I., Peters, M. E., and Cohan, A. (2020). Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Ben Noach, M. and Goldberg, Y. (2020). Compressing pre-trained language models by matrix decomposition. In Wong, K.-F., Knight, K., and Wu, H., editors, *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 884–889, Suzhou, China. Association for Computational Linguistics.

Bick, A., Li, K., Xing, E. P., Kolter, J. Z., and Gu, A. (2024). Transformers to SSMs: Distilling quadratic knowledge to subquadratic models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Biderman, S., Schoelkopf, H., Anthony, Q. G., Bradley, H., O'Brien, K., Hallahan, E., Khan, M. A., Purohit, S., Prashanth, U. S., Raff, E., et al. (2023). Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. (2020). PIQA: Reasoning about Physical Commonsense in Natural Language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7432–7439.

Bittermann, H. J. (1934). Elasticity of supply. *The American Economic Review*, pages 417–429.

Bojar, O., Buck, C., Federmann, C., Haddow, B., Koehn, P., Leveling, J., Monz, C., Pecina, P., Post, M., Saint-Amand, H., Soricut, R., Specia, L., and Tamchyna, A. (2014). Findings of the 2014 workshop on statistical machine translation. In Bojar, O., Buck, C., Federmann, C., Haddow, B., Koehn, P., Monz, C., Post, M., and Specia, L., editors, *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA. Association for Computational Linguistics.

Boo, Y. and Sung, W. (2020). Fixed-point optimization of transformer neural network. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1753–1757. IEEE.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J. D., Chen, D., and Dao, T. (2024). Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Forty-first International Conference on Machine Learning*.

Chen, T., Frankle, J., Chang, S., Liu, S., Zhang, Y., Wang, Z., and Carbin, M. (2020). The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems*, 33:15834–15846.

Chen, Y., Wang, G., Shang, J., Cui, S., Zhang, Z., Liu, T., Wang, S., Sun, Y., Yu, D., and Wu, H. (2024). NACL: A general and effective KV cache eviction framework for LLM at inference time. In Ku, L.-W., Martins, A., and Srikumar, V., editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7913–7926, Bangkok, Thailand. Association for Computational Linguistics.

Chiang, W.-L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J. E., et al. (2023). Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*.

Child, R., Gray, S., Radford, A., and Sutskever, I. (2019). Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder–decoder approaches. In Wu, D., Carpuat, M., Carreras, X., and Vecchi, E. M., editors, *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar. Association for Computational Linguistics.

Choi, S. (2024). Cross-architecture transfer learning for linear-cost inference transformers. *arXiv preprint arXiv:2404.02684*.

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J., and Weller, A. (2021). Rethinking attention with performers. In *International Conference on Learning Representations*.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. (2022). Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Chung, H. W., Fevry, T., Tsai, H., Johnson, M., and Ruder, S. (2021). Rethinking embedding coupling in pre-trained language models. In *International Conference on Learning Representations*.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. (2019). BoolQ: Exploring the surprising difficulty of natural yes/no questions. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, Minneapolis, Minnesota. Association for Computational Linguistics.

Clark, J. H., Garrette, D., Turc, I., and Wieting, J. (2022a). Canine: Pre-training an Efficient Tokenization-Free Encoder for Language Representation. *Transactions of the Association for Computational Linguistics*, 10:73–91.

Clark, J. H., Garrette, D., Turc, I., and Wieting, J. (2022b). Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91.

Clark, K., Luong, M.-T., Le, Q. V., and Manning, C. D. (2020). ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR*.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. (2018). Think you have solved question answering? Try arc, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., and Stoyanov, V. (2020). Unsupervised cross-lingual representation learning at scale. In Jurafsky, D., Chai, J., Schluter, N., and Tetreault, J., editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q., and Salakhutdinov, R. (2019). Transformer-XL: Attentive language models beyond a fixed-length context. In Korhonen, A., Traum, D., and Màrquez, L., editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. (2022). Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

Dao, T. and Gu, A. (2024). Transformers are SSMs: generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, pages 10041–10071.

Deiseroth, B., Brack, M., Schramowski, P., Kersting, K., and Weinbach, S. (2024). T-FREE: Subword tokenizer-free generative LLMs via sparse representations for memory-efficient embeddings. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N., editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 21829–21851, Miami, Florida, USA. Association for Computational Linguistics.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. (2022). GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

Dettmers, T., Svirschevski, R. A., Egiazarian, V., Kuznedelev, D., Frantar, E., Ashkboos, S., Borzunov, A., Hoefler, T., and Alistarh, D. (2024). SpQR: A sparse-quantized representation for near-lossless LLM weight compression. In *The Twelfth International Conference on Learning Representations*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Devoto, A., Zhao, Y., Scardapane, S., and Minervini, P. (2024). A simple and effective l_2 norm-based strategy for KV cache compression. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N., editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18476–18499, Miami, Florida, USA. Association for Computational Linguistics.

Dong, X., Fu, Y., Diao, S., Byeon, W., Chen, Z., Mahabaleshwarkar, A. S., Liu, S.-Y., Van Keirsbilck, M., Chen, M.-H., Suhara, Y., et al. (2024). Hymba: A hybrid-head architecture for small language models. *arXiv preprint arXiv:2411.13676*.

El Boukkouri, H., Ferret, O., Lavergne, T., Noji, H., Zweigenbaum, P., and Tsujii, J. (2020). CharacterBERT: Reconciling ELMo and BERT for word-level open-vocabulary representations from characters. In Scott, D., Bel, N., and Zong, C., editors, *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6903–6915, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Elfwing, S., Uchibe, E., and Doya, K. (2018). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11.

Fan, A., Grave, E., and Joulin, A. (2019). Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*.

Frantar, E. and Alistarh, D. (2023). Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning*, pages 10323–10337. PMLR.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. (2023). OPTQ: Accurate quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*.

Ganchev, K. and Dredze, M. (2008). Small statistical models by random feature mixing. In Rosario, B. and Paek, T., editors, *Proceedings of the ACL-08: HLT Workshop on Mobile Language Processing*, pages 19–20, Columbus, Ohio. Association for Computational Linguistics.

Ganesh, P., Chen, Y., Lou, X., Khan, M. A., Yang, Y., Sajjad, H., Nakov, P., Chen, D., and Winslett, M. (2021). Compressing large-scale transformer-based models: A case study on BERT. *Transactions of the Association for Computational Linguistics*, 9:1061–1080.

Gao, L., Tow, J., Abbasi, B., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., Le Noac'h, A., Li, H., McDonell, K., Muennighoff, N., Ociepa, C., Phang, J., Reynolds, L., Schoelkopf, H., Skowron, A., Sutawika, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. (2024). A framework for few-shot language model evaluation.

Gibbs, J. W. (1902). *Elementary principles in statistical mechanics: Developed with especial reference to the rational foundations of thermodynamics*. C. Scribner's sons.

Glorioso, P., Anthony, Q., Tokpanov, Y., Whittington, J., Pilault, J., Ibrahim, A., and Millidge, B. (2024). Zamba: A compact 7B SSM hybrid model. *arXiv preprint arXiv:2405.16712*.

Gordon, M., Duh, K., and Andrews, N. (2020). Compressing BERT: Studying the effects of weight pruning on transfer learning. In Gella, S., Welbl, J., Rei, M., Petroni, F., Lewis, P., Strubell, E., Seo, M., and Hajishirzi, H., editors, *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 143–155, Online. Association for Computational Linguistics.

Gritta, M., Xue, H., and Lampouras, G. (2025). Dresd: Dense retrieval for speculative decoding. *arXiv preprint arXiv:2502.15572*.

Gromov, A., Tirumala, K., Shapourian, H., Glorioso, P., and Roberts, D. (2025). The unreasonable ineffectiveness of the deeper layers. In *The Thirteenth International Conference on Learning Representations*.

Gu, A. and Dao, T. (2023). Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Gu, A., Goel, K., and Re, C. (2022). Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*.

Gu, Y., Dong, L., Wei, F., and Huang, M. (2024). MiniLLM: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations*.

Han, C., Wang, Q., Peng, H., Xiong, W., Chen, Y., Ji, H., and Wang, S. (2024). LM-infinite: Zero-shot extreme length generalization for large language models. In Duh, K., Gomez, H., and Bethard, S., editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3991–4008, Mexico City, Mexico. Association for Computational Linguistics.

He, Z., Yu, H., Gong, Z., Liu, S., Li, J., and Lin, W. (2025). Rodimus*: Breaking the accuracy-efficiency trade-off with efficient attentions. In *The Thirteenth International Conference on Learning Representations*.

He, Z., Zhong, Z., Cai, T., Lee, J., and He, D. (2024). REST: Retrieval-based speculative decoding. In Duh, K., Gomez, H., and Bethard, S., editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1582–1595, Mexico City, Mexico. Association for Computational Linguistics.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Vinyals, O., Rae, J. W., and Sifre, L. (2022). Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Hsieh, C.-Y., Li, C.-L., Yeh, C.-k., Nakhost, H., Fujii, Y., Ratner, A., Krishna, R., Lee, C.-Y., and Pfister, T. (2023). Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In Rogers, A., Boyd-Graber, J., and Okazaki, N., editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8003–8017, Toronto, Canada. Association for Computational Linguistics.

Hu, E. J., yelong shen, Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. (2022). LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2017). Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898.

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., and Liu, Q. (2020). TinyBERT: Distilling BERT for natural language understanding. In Cohn, T., He, Y., and Liu, Y., editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics.

Jurafsky, D. and Martin, J. H. (2025). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd edition*. Online manuscript released January 12, 2025.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Kasai, J., Peng, H., Zhang, Y., Yogatama, D., Ilharco, G., Pappas, N., Mao, Y., Chen, W., and Smith, N. A. (2021). Finetuning pretrained transformers into RNNs. In Moens, M.-F., Huang, X., Specia, L., and Yih, S. W.-t., editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10630–10643, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. (2020). Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR.

Khetan, A. and Karnin, Z. (2020). schuBERT: Optimizing elements of BERT. In Jurafsky, D., Chai, J., Schluter, N., and Tetreault, J., editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2807–2818, Online. Association for Computational Linguistics.

Kitaev, N., Kaiser, L., and Levskaya, A. (2020). Reformer: The efficient transformer. In *International Conference on Learning Representations*.

Korthikanti, V. A., Casper, J., Lym, S., McAfee, L., Andersch, M., Shoeybi, M., and Catanzaro, B. (2023). Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5:341–353.

Koto, F., Lau, J. H., and Baldwin, T. (2021). Discourse probing of pretrained language models. In Toutanova, K., Rumshisky, A., Zettlemoyer, L., Hakkani-Tur, D., Beltagy, I., Bethard, S., Cotterell, R., Chakraborty, T., and Zhou, Y., editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3849–3864, Online. Association for Computational Linguistics.

Kudo, T. and Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In Blanco, E. and Lu, W., editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Lacoste, A., Luccioni, A., Schmidt, V., and Dandres, T. (2019). Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2020a). Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2020b). Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.

Lee-Thorp, J., Ainslie, J., Eckstein, I., and Ontanon, S. (2022). FNet: Mixing tokens with Fourier transforms. In Carpuat, M., de Marneffe, M.-C., and Meza Ruiz, I. V., editors, *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4296–4313, Seattle, United States. Association for Computational Linguistics.

Lenz, B., Lieber, O., Arazi, A., Bergman, A., Manevich, A., Peleg, B., Aviram, B., Almagor, C., Fridman, C., Padnos, D., Gissin, D., Jannai, D., Muhlgay, D., Zimberg, D., Gerber, E. M., Dolev, E., Krakovsky, E., Safahi, E., Schwartz, E., Cohen, G., Shachaf, G., Rozenblum, H., Bata, H., Blass, I., Magar, I., Dalmedigos, I., Osin, J., Fadlon, J., Rozman, M., Danos, M., Gokhman, M., Zusman, M., Gidron, N., Ratner, N., Gat, N., Rozen, N., Fried, O., Leshno, O., Antverg, O., Abend, O., Dagan, O., Cohavi, O., Alon, R., Belson, R., Cohen, R., Gilad, R., Glozman, R., Lev, S., Shalev-Shwartz, S., Meirom, S. H., Delbari, T., Ness, T., Asida, T., Gal, T. B., Braude, T., Pumerantz, U., Cohen, J., Belinkov, Y., Globerson, Y., Levy, Y. P., and Shoham, Y. (2025). Jamba: Hybrid Transformer-Mamba language models. In *The Thirteenth International Conference on Learning Representations*.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2020). BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Jurafsky, D., Chai, J., Schluter, N., and Tetreault, J., editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Lhoest, Q., Villanova del Moral, A., Jernite, Y., Thakur, A., von Platen, P., Patil, S., Chaumond, J., Drame, M., Plu, J., Tunstall, L., Davison, J., Šaško, M., Chhablani, G., Malik, B., Brandeis, S., Le Scao, T., Sanh, V., Xu, C., Patry, N., McMillan-Major, A., Schmid, P., Gugger, S., Delangue, C., Matussière, T., Debut, L., Bekman, S., Cistac, P., Goehringer, T., Mustar, V., Lagunas, F., Rush, A., and Wolf, T. (2021). Datasets: A community library for natural language processing. In Adel, H. and Shi, S., editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Li, B., Kong, Z., Zhang, T., Li, J., Li, Z., Liu, H., and Ding, C. (2020a). Efficient transformer-based large scale language representations using hardware-friendly block structured pruning. In Cohn, T., He, Y., and Liu, Y., editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3187–3199, Online. Association for Computational Linguistics.

Li, C., Farkhoor, H., Liu, R., and Yosinski, J. (2018). Measuring the intrinsic dimension of objective landscapes. In *International Conference on Learning Representations*.

Li, J., Liu, X., Zhao, H., Xu, R., Yang, M., and Jin, Y. (2020b). BERT-EMD: Many-to-many layer mapping for BERT compression with earth mover‘s distance. In Webber, B., Cohn, T., He, Y., and Liu, Y., editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3009–3018, Online. Association for Computational Linguistics.

Li, L., Lin, Y., Ren, S., Li, P., Zhou, J., and Sun, X. (2021). Dynamic knowledge distillation for pre-trained language models. In Moens, M.-F., Huang, X., Specia, L., and Yih, S. W.-t., editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Li, Y., Bai, R., and Huang, H. (2024a). Spin glass model of in-context learning. *arXiv preprint arXiv:2408.02288*.

Li, Y., Wei, F., Zhang, C., and Zhang, H. (2024b). Eagle: Speculative sampling requires

rethinking feature uncertainty. In *International Conference on Machine Learning*, pages 28935–28948. PMLR.

Liang, D., Gonen, H., Mao, Y., Hou, R., Goyal, N., Ghazvininejad, M., Zettlemoyer, L., and Khabsa, M. (2023). XLM-V: Overcoming the vocabulary bottleneck in multilingual masked language models. In Bouamor, H., Pino, J., and Bali, K., editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13142–13152, Singapore. Association for Computational Linguistics.

Lin, Z., Liu, J., Yang, Z., Hua, N., and Roth, D. (2020). Pruning redundant mappings in transformer models via spectral-normalized identity prior. In Cohn, T., He, Y., and Liu, Y., editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 719–730, Online. Association for Computational Linguistics.

Lin, Z., Nikishin, E., He, X., and Courville, A. (2025). Forgetting transformer: Softmax attention with a forget gate. In *The Thirteenth International Conference on Learning Representations*.

Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al. (2024a). Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Liu, H., Tam, D., Mohammed, M., Mohta, J., Huang, T., Bansal, M., and Raffel, C. (2022). Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

Liu, M., Rabbani, T., O'Halloran, T., Sankaralingam, A., Hartley, M.-A., Huang, F., Fermüller, C., and Aloimonos, Y. (2024b). Hashevict: A pre-attention kv cache eviction strategy using locality-sensitive hashing. *arXiv preprint arXiv:2412.16187*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

Liu, Z., Desai, A., Liao, F., Wang, W., Xie, V., Xu, Z., Kyrillidis, A., and Shrivastava, A. (2023). Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364.

Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Lozhkov, A., Ben Allal, L., von Werra, L., and Wolf, T. (2024). FineWeb-Edu: The finest collection of educational content .

Ma, S., Wang, H., Huang, S., Zhang, X., Hu, Y., Song, T., Xia, Y., and Wei, F. (2025). Bitnet b1. 58 2b4t technical report. *arXiv preprint arXiv:2504.12285*.

Ma, S., Wang, H., Ma, L., Wang, L., Wang, W., Huang, S., Dong, L., Wang, R., Xue, J., and Wei, F. (2024). The era of 1-bit llms: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764*, 1(4).

Mamou, J., Pereg, O., Korat, D., Berchansky, M., Timor, N., Wasserblat, M., and Schwartz, R. (2024). Accelerating speculative decoding using dynamic speculation length. *CoRR*.

Mao, H. H. (2022). Fine-tuning pre-trained transformers into decaying fast weights. In Goldberg, Y., Kozareva, Z., and Zhang, Y., editors, *Proceedings of the 2022 Conference on Empirical*

Methods in Natural Language Processing, pages 10236–10242, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Mao, Y., Wang, Y., Wu, C., Zhang, C., Wang, Y., Zhang, Q., Yang, Y., Tong, Y., and Bai, J. (2020). LadaBERT: Lightweight adaptation of BERT through hybrid model compression. In Scott, D., Bel, N., and Zong, C., editors, *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3225–3234, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Mercat, J., Vasiljevic, I., Keh, S. S., Arora, K., Dave, A., Gaidon, A., and Kollar, T. (2024). Linearizing large language models. In *First Conference on Language Modeling*.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2017). Pointer sentinel mixture models. In *International Conference on Learning Representations*.

Miao, X., Oliaro, G., Zhang, Z., Cheng, X., Wang, Z., Zhang, Z., Wong, R. Y. Y., Zhu, A., Yang, L., Shi, X., et al. (2024). Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 932–949.

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., et al. (2021). Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 1–15.

Naumov, M., Mudigere, D., Shi, H.-J. M., Huang, J., Sundaraman, N., Park, J., Wang, X., Gupta, U., Wu, C.-J., Azzolini, A. G., et al. (2019). Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091*.

Nawrot, P., Chorowski, J., Lancucki, A., and Ponti, E. M. (2023). Efficient transformers with dynamic token pooling. In Rogers, A., Boyd-Graber, J., and Okazaki, N., editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6403–6417, Toronto, Canada. Association for Computational Linguistics.

Neyshabur, B., Tomioka, R., Salakhutdinov, R., and Srebro, N. (2017). Geometry of optimization and implicit regularization in deep learning. *arXiv preprint arXiv:1705.03071*.

Nielsen, J., Schneider-Kamp, P., and Galke, L. (2025). Continual quantization-aware pre-training: When to transition from 16-bit to 1.58-bit pre-training for bitnet language models? *arXiv preprint arXiv:2502.11895*.

Orvieto, A., Smith, S. L., Gu, A., Fernando, A., Gulcehre, C., Pascanu, R., and De, S. (2023). Resurrecting recurrent neural networks for long sequences. In *International Conference on Machine Learning*, pages 26670–26698. PMLR.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Gray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., and Lowe, R. (2022). Training language models to follow instructions with human feedback. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

Ouyang, X., Ge, T., Hartvigsen, T., Zhang, Z., Mi, H., and Yu, D. (2025). Low-bit quantization favors undertrained LLMs. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T., editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32338–32348, Vienna, Austria. Association for Computational Linguistics.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Biderman, S., Cao, H., Cheng, X., Chung, M., Derczynski, L., Du, X., Grella, M., Gv, K., He, X., Hou, H., Kazienko, P., Kocon, J., Kong, J., Koptyra, B., Lau, H., Lin, J., Mantri, K. S. I., Mom, F., Saito, A., Song, G., Tang, X., Wind, J., Woźniak, S., Zhang, Z., Zhou, Q., Zhu, J., and Zhu, R.-J. (2023). RWKV: Reinventing RNNs for the transformer era. In Bouamor, H., Pino, J., and Bali, K., editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14048–14077, Singapore. Association for Computational Linguistics.

Peng, B., Goldstein, D., Anthony, Q. G., Albalak, A., Alcaide, E., Biderman, S., Cheah, E., Ferdinan, T., GV, K. K., Hou, H., Krishna, S., Jr., R. M., Muennighoff, N., Obeid, F., Saito, A., Song, G., Tu, H., Zhang, R., Zhao, B., Zhao, Q., Zhu, J., and Zhu, R.-J. (2024). Eagle and Finch: RWKV with matrix-valued states and dynamic recurrence. In *First Conference on Language Modeling*.

Peng, B., Zhang, R., Goldstein, D., Alcaide, E., Hou, H., Lu, J., Merrill, W., Song, G., Tan, K., Utpala, S., et al. (2025). RWKV-7" Goose" with expressive dynamic state evolution. *arXiv preprint arXiv:2503.14456*.

Peng, D. and Cao, H. (2024). E-Tamba: Efficient Transformer-Mamba layer transplantation. In *NeurIPS 2024 Workshop on Fine-Tuning in Modern Machine Learning: Principles and Scalability*.

Peng, H., Pappas, N., Yogatama, D., Schwartz, R., Smith, N., and Kong, L. (2021). Random feature attention. In *International Conference on Learning Representations*.

Poli, M., Massaroli, S., Nguyen, E., Fu, D. Y., Dao, T., Baccus, S., Bengio, Y., Ermon, S., and Ré, C. (2023). Hyena hierarchy: Towards larger convolutional language models. In *International Conference on Machine Learning*, pages 28043–28078. PMLR.

Prakash, P., Shashidhar, S. K., Zhao, W., Rongali, S., Khan, H., and Kayser, M. (2020). Compressing transformer-based semantic parsing models using compositional code embeddings. In Cohn, T., He, Y., and Liu, Y., editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4711–4717, Online. Association for Computational Linguistics.

Prasanna, S., Rogers, A., and Rumshisky, A. (2020). When BERT Plays the Lottery, All Tickets Are Winning. In Webber, B., Cohn, T., He, Y., and Liu, Y., editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3208–3229, Online. Association for Computational Linguistics.

Pruthi, D., Dhingra, B., and Lipton, Z. C. (2019). Combating adversarial misspellings with robust word recognition. In Korhonen, A., Traum, D., and Màrquez, L., editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5582–5591, Florence, Italy. Association for Computational Linguistics.

Qin, Z., Sun, W., Deng, H., Li, D., Wei, Y., Lv, B., Yan, J., Kong, L., and Zhong, Y. (2022). cosformer: Rethinking softmax in attention. In *International Conference on Learning Representations*.

Qin, Z., Yang, S., Sun, W., Shen, X., Li, D., Sun, W., and Zhong, Y. (2024). HGRN2: Gated linear RNNs with state expansion. In *First Conference on Language Modeling*.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Rajabzadeh, H., Jafari, A., Sharma, A., Jami, B., Kwon, H. J. H., Ghodsi, A., Chen, B., and Rezagholizadeh, M. (2024). Echoatt: Attend, copy, then adjust for more efficient large language models. In *NeurIPS Efficient Natural Language and Speech Processing Workshop*, pages 259–269. PMLR.

Rajaraman, A. and Ullman, J. D. (2011). *Mining of massive datasets*. Cambridge University Press.

Rajpurkar, P., Jia, R., and Liang, P. (2018). Know what you don't know: Unanswerable questions for SQuAD. In Gurevych, I. and Miyao, Y., editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). SQuAD: 100,000+ questions for machine comprehension of text. In Su, J., Duh, K., and Carreras, X., editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Ravi, S. (2019). Efficient on-device models using neural projections. In *International Conference on Machine Learning*, pages 5370–5379. PMLR.

Ren, W., Ma, W., Yang, H., Wei, C., Zhang, G., and Chen, W. (2025). Vamba: Understanding hour-long videos with hybrid mamba-transformers. *arXiv preprint arXiv:2503.11579*.

Rivest, R. and Dusse, S. (1992). The MD5 message-digest algorithm.

Roemmele, M., Bejan, C. A., and Gordon, A. S. (2011). Choice of Plausible Alternatives: An Evaluation of Commonsense Causal Reasoning. In *AAAI spring symposium: logical formalizations of commonsense reasoning*, pages 90–95.

Sanh, V., Wolf, T., and Rush, A. (2020). Movement pruning: Adaptive sparsity by fine-tuning. *Advances in Neural Information Processing Systems*, 33:20378–20389.

Sankar, C., Ravi, S., and Kozareva, Z. (2021a). On-device text representations robust to misspellings via projections. In Merlo, P., Tiedemann, J., and Tsarfaty, R., editors, *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2871–2876, Online. Association for Computational Linguistics.

Sankar, C., Ravi, S., and Kozareva, Z. (2021b). ProFormer: Towards on-device LSH projection based transformers. In Merlo, P., Tiedemann, J., and Tsarfaty, R., editors, *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2823–2828, Online. Association for Computational Linguistics.

Schlag, I., Irie, K., and Schmidhuber, J. (2021). Linear transformers are secretly fast weight programmers. In *International conference on machine learning*, pages 9355–9366. PMLR.

Schuster, T., Fisch, A., Jaakkola, T., and Barzilay, R. (2021). Consistent accelerated inference via confident adaptive transformers. In Moens, M.-F., Huang, X., Specia, L., and Yih, S. W.-t., editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4962–4979, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. (2020). Green AI. *Communications of the ACM*, 63(12):54–63.

Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In Erk, K. and Smith, N. A., editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Shazeer, N. (2019). Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*.

Shen, S., Dong, Z., Ye, J., Ma, L., Yao, Z., Gholami, A., Mahoney, M. W., and Keutzer, K. (2020). Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821.

Shi, Q., Petterson, J., Dror, G., Langford, J., Smola, A., Strehl, A., and Vishwanathan, S. (2009). Hash kernels. In *Artificial intelligence and statistics*, pages 496–503. PMLR.

Shu, R. and Nakayama, H. (2018). Compressing word embeddings via deep compositional code learning. In *International Conference on Learning Representations*.

Siems, J., Carstensen, T., Zela, A., Hutter, F., Pontil, M., and Grazzi, R. (2025). DeltaProduct: Improving state-tracking in linear RNNs via Householder products. *arXiv preprint arXiv:2502.10297*.

Singh, A. (2025). Meta Llama 4: The future of multimodal AI. *Available at SSRN 5208228*.

Smith, S., Patwary, M., Norick, B., LeGresley, P., Rajbhandari, S., Casper, J., Liu, Z., Prabhumoye, S., Zerveas, G., Korthikanti, V., et al. (2022). Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*.

Soboleva, D., Al-Khateeb, F., Myers, R., Steeves, J. R., Hestness, J., and Dey, N. (2023). SlimPajama: A 627B token cleaned and deduplicated version of RedPajama.

Stock, P., Fan, A., Graham, B., Grave, E., Gribonval, R., Jegou, H., and Joulin, A. (2020). Training with quantization noise for extreme model compression. In *International Conference on Learning Representations*.

Strubell, E., Ganesh, A., and McCallum, A. (2019a). Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650.

Strubell, E., Ganesh, A., and McCallum, A. (2019b). Energy and policy considerations for deep learning in NLP. In Korhonen, A., Traum, D., and Màrquez, L., editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., and Liu, Y. (2021). Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*.

Sun, L., Hashimoto, K., Yin, W., Asai, A., Li, J., Yu, P., and Xiong, C. (2020a). Adv-BERT: BERT is not robust on misspellings! Generating nature adversarial samples on BERT. *arXiv preprint arXiv:2003.04985*.

Sun, M., Liu, Z., Bair, A., and Kolter, J. Z. (2024). A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*.

Sun, S., Cheng, Y., Gan, Z., and Liu, J. (2019). Patient knowledge distillation for BERT model compression. In Inui, K., Jiang, J., Ng, V., and Wan, X., editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4323–4332, Hong Kong, China. Association for Computational Linguistics.

Sun, Z., Yu, H., Song, X., Liu, R., Yang, Y., and Zhou, D. (2020b). MobileBERT: a compact task-agnostic BERT for resource-limited devices. In Jurafsky, D., Chai, J., Schluter, N., and Tetreault, J., editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2158–2170, Online. Association for Computational Linguistics.

Svenstrup, D., Hansen, J., and Winther, O. (2017). Hash embeddings for efficient word representations. *Advances in neural information processing systems*, 30.

Tahaei, M., Charlaix, E., Nia, V., Ghodsi, A., and Rezagholizadeh, M. (2022). KroneckerBERT: Significant compression of pre-trained language models through kronecker decomposition and knowledge distillation. In Carpuat, M., de Marneffe, M.-C., and Meza Ruiz, I. V., editors, *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2116–2127, Seattle, United States. Association for Computational Linguistics.

Tambe, T., Hooper, C., Pentecost, L., Jia, T., Yang, E.-Y., Donato, M., Sanh, V., Whatmough, P., Rush, A. M., Brooks, D., et al. (2021). Edgebert: Sentence-level energy optimizations for latency-aware multi-task nlp inference. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 830–844.

Tao, C., Hou, L., Zhang, W., Shang, L., Jiang, X., Liu, Q., Luo, P., and Wong, N. (2022). Compression of generative pre-trained language models via quantization. In Muresan, S., Nakov, P., and Villavicencio, A., editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4821–4836, Dublin, Ireland. Association for Computational Linguistics.

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. (2023). Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

Tay, Y., Bahri, D., Metzler, D., Juan, D.-C., Zhao, Z., and Zheng, C. (2021). Synthesizer: Rethinking self-attention for transformer models. In *International conference on machine learning*, pages 10183–10192. PMLR.

Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. (2022a). Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28.

Tay, Y., Tran, V. Q., Ruder, S., Gupta, J., Chung, H. W., Bahri, D., Qin, Z., Baumgartner, S.,

Yu, C., and Metzler, D. (2022b). Charformer: Fast character transformers via gradient-based subword tokenization. In *International Conference on Learning Representations*.

Tay, Y., Zhang, A., Luu, A. T., Rao, J., Zhang, S., Wang, S., Fu, J., and Hui, S. C. (2019). Lightweight and efficient neural natural language processing with quaternion networks. In Korhonen, A., Traum, D., and Màrquez, L., editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1494–1503, Florence, Italy. Association for Computational Linguistics.

Team, J., Lenz, B., Arazi, A., Bergman, A., Manevich, A., Peleg, B., Aviram, B., Almagor, C., Fridman, C., Padnos, D., et al. (2024). Jamba-1.5: Hybrid transformer-mamba models at scale. *arXiv preprint arXiv:2408.12570*.

Tolstikhin, I. O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Steiner, A., Keysers, D., Uszkoreit, J., et al. (2021). MLP-Mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34:24261–24272.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Varshney, N., Chatterjee, A., Parmar, M., and Baral, C. (2024). Investigating acceleration of LLaMA inference by enabling intermediate layer decoding via instruction tuning with 'LITE'. In Duh, K., Gomez, H., and Bethard, S., editors, *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 3656–3677, Mexico City, Mexico. Association for Computational Linguistics.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Vulić, I., Ponti, E. M., Litschko, R., Glavaš, G., and Korhonen, A. (2020). Probing pretrained language models for lexical semantics. In Webber, B., Cohn, T., He, Y., and Liu, Y., editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7222–7240, Online. Association for Computational Linguistics.

Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. (2019). Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. (2018). GLUE: A multi-task benchmark and analysis platform for natural language understanding. In Linzen, T., Chrupała, G., and Alishahi, A., editors, *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Wang, B., Ren, Y., Shang, L., Jiang, X., and Liu, Q. (2022). Exploring extreme parameter compression for pre-trained language models. In *International Conference on Learning Representations*.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. (2020). Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

Weidinger, L., Uesato, J., Rauh, M., Griffin, C., Huang, P.-S., Mellor, J., Glaese, A., Cheng, M.,

Balle, B., Kasirzadeh, A., et al. (2022). Taxonomy of risks posed by language models. In *2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 214–229.

Welbl, J., Liu, N. F., and Gardner, M. (2017). Crowdsourcing multiple choice science questions. In Derczynski, L., Xu, W., Ritter, A., and Baldwin, T., editors, *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 94–106, Copenhagen, Denmark. Association for Computational Linguistics.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. (2023). Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pages 38087–38099. PMLR.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. (2024). Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*.

Xiao, T., Li, Y., Zhu, J., Yu, Z., and Liu, T. (2019). Sharing attention weights for fast transformer. *arXiv preprint arXiv:1906.11024*.

Xue, H. and Aletras, N. (2022). HashFormers: Towards vocabulary-independent pre-trained transformers. In Goldberg, Y., Kozareva, Z., and Zhang, Y., editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 7862–7874, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Xue, H. and Aletras, N. (2023). Pit one against many: Leveraging attention-head embeddings for parameter-efficient multi-head attention. In Bouamor, H., Pino, J., and Bali, K., editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10355–10373, Singapore. Association for Computational Linguistics.

Xue, L., Barua, A., Constant, N., Al-Rfou, R., Narang, S., Kale, M., Roberts, A., and Raffel, C. (2022). ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306.

Yamaguchi, A., Chrysostomou, G., Margatina, K., and Aletras, N. (2021). Frustratingly simple pretraining alternatives to masked language modeling. In Moens, M.-F., Huang, X., Specia, L., and Yih, S. W.-t., editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3116–3125, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yan, Y., Chen, J., Qi, W., Bhendawade, N., Gong, Y., Duan, N., and Zhang, R. (2021). El-attention: Memory efficient lossless attention for generation. In *International Conference on Machine Learning*, pages 11648–11658. PMLR.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al. (2025a). Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., et al. (2024a). Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Yang, S., Kautz, J., and Hatamizadeh, A. (2025b). Gated delta networks: Improving mamba2 with delta rule. In *The Thirteenth International Conference on Learning Representations*.

Yang, S., Wang, B., Zhang, Y., Shen, Y., and Kim, Y. (2024b). Parallelizing linear transformers with the delta rule over sequence length. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.

Yu, W., Luo, M., Zhou, P., Si, C., Zhou, Y., Wang, X., Feng, J., and Yan, S. (2022). Metaformer is actually what you need for vision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10819–10829.

Zadeh, A. H., Edo, I., Awad, O. M., and Moshovos, A. (2020). Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 811–824. IEEE.

Zafrir, O., Boudoukh, G., Izsak, P., and Wasserblat, M. (2019). Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pages 36–39. IEEE.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. (2020). Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. (2019). HellaSwag: Can a machine really finish your sentence? In Korhonen, A., Traum, D., and Màrquez, L., editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy. Association for Computational Linguistics.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2022). Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. (2023). H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

Zhao, S., Gupta, R., Song, Y., and Zhou, D. (2021). Extremely small BERT models from mixed-vocabulary training. In Merlo, P., Tiedemann, J., and Tsarfaty, R., editors, *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2753–2759, Online. Association for Computational Linguistics.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27.

Zimmer, M., Gritta, M., Lampouras, G., Ammar, H. B., and Wang, J. (2025). Mixture of attentions for speculative decoding. In *The Thirteenth International Conference on Learning Representations*.