University of Warsaw
Faculty of Mathematics, Informatics, and Mechanics

Sebastian Jaszczur

Efficient Large Language Models with Conditional Computation

PhD thesis
in Computer Science

Supervisor:
Marek Cygan
Institute of Informatics
University of Warsaw

Warsaw, September 2024

Supervisor's Statement

I confirm that the presented thesis was prepared under my supervision and that it fulfills the requirements for the doctoral degree in the field of Natural Sciences, in the discipline of Computer Science.

Date Supervisor's Signature

Author's Statement

I declare that the presented thesis was prepared by me and that none of its content was obtained through unlawful means.

The thesis has never before been subject to any procedure for obtaining an academic degree. Furthermore, I declare that the presented version of the thesis is identical to the attached electronic version.

Date Author's Signature

Oświadczenie kierującego pracą

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem i stwierdzam, że spełnia ona warunki do przedstawienia jej w postępowaniu o nadanie stopnia doktora w dziedzinie nauk ścisłych i przyrodniczych w dyscyplinie informatyka.

Data Podpis kierującego pracą

Oświadczenie autora pracy

Oświadczam, że niniejsza rozprawa doktorska została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem stopnia doktora w innej jednostce. Niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora pracy

Abstract

This PhD thesis examines the potential of improving the efficiency of language models by incorporating Conditional Computation into the architecture. Deep Learning methods have taken over the Natural Language Processing (NLP) field in recent years. Particularly after the invention of the Transformer architecture, these models have been scaled to unprecedented sizes. Research works, such as scaling laws, highlight the importance of the sheer number of parameters and the size of the dataset. It is evident that this trend of training larger and larger models will only continue. This is where Conditional Computation can provide the needed efficiency for training and running Large Language Models (LLMs).

First, after describing the background of language models and Conditional Computation, we introduce sparse variants of standard Transformer layers, which enable considerable speedups during inference without impacting model quality. Then, we present a fine-grained Mixture of Experts (MoE), where the granularity of experts is parametrized. We also design new scaling laws, taking the new parameter into account and showing the improved efficiency of granular MoE. Furthermore, we correct claims made by prior works about the relative efficiency of MoE when scaling the model size.

We then present Mixture of Tokens, a novel variant of Mixture of Experts with a fully continuous architecture enabling more stable training. This design is further strengthened by the introduction of transition tuning, showing that a Mixture of Tokens can be converted to a sparse Mixture of Experts during the fine-tuning phase. Subsequently, we introduce SPLiCe, which uses fine-tuning to improve the efficiency of long-context utilization, enabling better performance during inference on long documents. Finally, we incorporate Mixture of Experts into the recently developed Mamba architecture, demonstrating the strength of combining state space models with Conditional Computation.

In summary, this thesis introduces multiple improvements to Conditional Computation, enhancing the efficiency of Large Language Models. It demonstrates the strength and resiliency of these techniques across different scales and architectures. The work not only advances the field with new insights but also challenges existing ideas and corrects previous research. With multiple recent Large Language Models using different flavors of Mixture of Experts, it is clear that Conditional Computation is likely to stay in language modeling for a long time.

Keywords

Deep Learning, Deep Neural Networks, Large Language Models, Transformer, Conditional Computation, Mixture of Experts

Tytuł pracy w języku polskim

Efektywne duże modele językowe przy użyciu obliczeń warunkowych

Streszczenie w języku polskim

Ta praca doktorska bada możliwości poprawy efektywności modeli językowych przy użyciu obliczeń warunkowych. W ostatnich latach głębokie sieci neuronowe zdominowały dziedzinę przetwarzania języka naturalnego (NLP). Szczególnie po publikacji Transformera, sieci neuronowe urosły do niespotykanych dotąd rozmiarów. Publikacje wprowadzające prawa skalowania pokazały istotną wagę liczby parametrów modelu oraz wielkości zbioru treningowego. Jest oczywiste, że z biegiem czasu będą trenowane coraz większe i większe modele. Obliczenia warunkowe mogą zapewnić tak potrzebną efektywność dla dużych modeli językowych (LLM).

Najpierw, po opisaniu obecnego stanu modeli językowych oraz obliczeń warunkowych, wprowadzamy warianty standardowych warstw Transformera, które znacząco przyspieszają inferencję modelu, bez pogorszenia jego jakości. Następnie przedstawiamy drobnoziarnistą wersję Mixture of Experts (MoE), w której granularność ekspertów jest parametryzowana. Wyprowadzamy również nowe prawa skalowania dla modeli językowych z MoE, które uwzględniają granularność oraz pokazują efektywność drobnoziarnistego MoE. Ponadto korygujemy twierdzenia wcześniejszych prac na temat skalowania MoE oraz ich względnej efektywności.

Następnie wprowadzamy Mixture of Tokens (MoT), wariant MoE z w pełni ciągłą i różniczkowalną architekturą, umożliwiając stabilniejszy trening. Ta technika jest dodatkowo wsparta wprowadzeniem specjalnej metody dotrenowywania, pokazując, że MoT może zostać przekształcone w standardowe MoE podczas fine-tuningu. Następnie prezentujemy SPLiCe, który wykorzystuje fine-tuning do poprawy efektywności wykorzystania długiego kontekstu, umożliwiając lepszą jakość inferencji na długich dokumentach. Na koniec, łączymy MoE z niedawno opracowaną architekturą Mamba, pokazując potencjał wprowadzania obliczeń warunkowych do modele SSM.

Podsumowując, ta praca wprowadza wiele usprawnień do obliczeń warunkowych, zwiększając efektywność dużych modeli językowych. Ponadto, pokazuje wartość i stabilność obliczeń warunkowych przy różnych wielkościach oraz architekturach modeli. Ta praca nie tylko posuwa dziedzinę naprzód przez nowe techniki, ale także kwestionuje istniejące przekonania oraz koryguje wcześniejsze badania. W ostatnim czasie wiele dużych modeli językowych używało różne odmiany MoE, i jasno widać, że obliczenia warunkowe pozostaną istotną częścią modeli językowych w przyszłości.

Słowa kluczowe

Głębokie sieci neuronowe, modele językowe, Transformer, oblicznia warunkowe, Mixture of Experts

Contents

Chapter 1

Introduction

Die Grenzen meiner Sprache bedeuten die Grenzen meiner Welt.
"The limits of my language mean the limits of my world."
— Ludwig Wittgenstein, *Tractatus Logico-Philosophicus*

1.1 Growing Importance of Efficient Language Models

This PhD thesis examines the potential of improving Large Language Models (LLMs) with Conditional Computation. Deep Learning methods have taken over the Natural Language Processing (NLP) field in recent years. Starting with the invention of the Transformer architecture (Vaswani et al., 2017), neural networks first replaced classical methods in NLP in fields like machine translation and language understanding. Then, they enabled the practical use of generative Artificial Intelligence through language modeling, which surged in popularity after the release of ChatGPT by OpenAI in November 2022. Every month, there are new developments in this space, with novel uses of LLMs made possible by their new abilities. Enumerating the current possibilities of LLMs is almost pointless since, by the time you read this, the state-of-the-art has likely advanced much further.

All of this progress was aided not only by algorithmic improvements in neural networks but also by a rapid increase in the scale of model and dataset sizes. These advancements have led to the plausible perspective of future significant research and commercial applications, which in turn fuel ever-increasing budgets for both research and industry projects in this area. Consequently, these increased budgets and efforts make further improvements even more frequent, forming a positive feedback loop.

Both training and inference of LLMs incur substantial costs. Strubell et al. (2019) estimate that training a single base BERT model cost around $4k-$12k - that was in 2018, with 110M parameters and 3.3B tokens in the training data (Devlin et al., 2018). In 2020, the training of GPT-3 (Brown et al., 2020) with 175B parameters on 300B tokens cost approximately $4.6M (Li, 2020). The training of GPT-4 (OpenAI, 2023), released in 2023, cost over $100M. For 2024, there are estimates that OpenAI is set to spend around $3B on training their models and an additional $4B on running inference (Barrabi, 2024). Anthropic's CEO suggested that LLMs may cost up to $10B to train by 2025 and up to $100B by 2027.

The exorbitant sizes of all state-of-the-art Large Language Models are integral to their success. This is also supported by research on scaling laws (Kaplan et al., 2020; Hoffmann et al., 2022), which suggests a predictable relationship between model size, dataset size, and the quality of the model's predictions. These findings imply that scaling

will remain a vital component in training increasingly capable models.

Therefore, it is easy to see that the most crucial bottleneck to improving Large Language Models is their cost. This thesis aims to enable the training of even larger and better models on a limited budget by increasing the efficiency of LLMs through the use of Conditional Computation.

1.2 Transformer and Large Language Models

A detailed description of the Transformer architecture, as introduced in Vaswani et al. (2017), can be found in textbooks such as Raschka (2024), Tunstall et al. (2022), François (2023), and numerous online resources. Here, we cover only the parts of the architecture relevant to this thesis, mainly the feed-forward layer, along with the notation used in the following chapters. While the attention layer is a crucial component of the Transformer, we do not describe it, as the inner workings of the attention mechanism are largely orthogonal to this work.

Unless specified otherwise, when we refer to Transformers, we mean exclusively decoder-only Transformers (like models from the GPT family). Although encoder-decoder and encoder-only architectures are still in use today, the simplicity and training efficiency[1] of the decoder-only architecture seem to be the deciding factors for the popularity of decoder-only models. Consequently, it is the architecture of choice for current Large Language Models.

A standard decoder-only Transformer (Radford et al., 2018, 2019; Kaplan et al., 2020; Brown et al., 2020) consists of an embedding layer, a stack of alternating attention and feed-forward (also called MLP) layers with normalization before them[2], and an unembedding layer (also called the prediction head or the final linear layer). See Figure 1.1. In the model, each input token is converted by the embedding layer into a vector of size d_{model}, the dimension maintained across all layers in the residual stream. Each layer refines the token representations by adding its output to them.

The standard feed-forward layer consists of two linear transformations and a non-linearity ϕ in between. It can be described as $\text{FF}(x) = \phi(xW_1 + b_1)W_2 + b_2$, with W_1 mapping from d_{model} to d_{ff}, and W_2 back to the original d_{model}. It is standard (Radford et al., 2018; Rae et al., 2022; Touvron et al., 2023; Jiang et al., 2023) to set the hidden dimension as $d_{\text{ff}} = 4 \cdot d_{\text{model}}$. Biases are often skipped, as at larger scales, they cause training instabilities while not improving the model much (Chowdhery et al., 2022). See Figure 1.2 for a diagram of a feed-forward layer with ReLU activation. While ReLU activation was traditionally used, it has been mostly replaced by SwiGLU. Although changing the activation function provides meaningful improvements to the model, this matter is largely orthogonal to the works presented in this thesis.

Feed-forward layers contain the majority of Transformer parameters and consume the majority of the Transformer's computational budget, measured in terms of FLOPs.

[1] Training efficiency may come from the simple information density of the language modeling objective in decoder-only models. While decoder-only models provide even 16 bits (for a vocabulary of 60k tokens) of information per token, the encoder-decoder model provides labels only for the latter half of tokens, achieving 50% efficiency compared to the decoder-only model. For encoder-only models using masked language modeling, only masked tokens receive this amount of information, corresponding to an efficiency of around 15%.

[2] In the original Transformer, normalization was done on the residual stream after addition, known as post-norm. However, all modern Transformers use pre-norm, normalizing just the input to each layer.

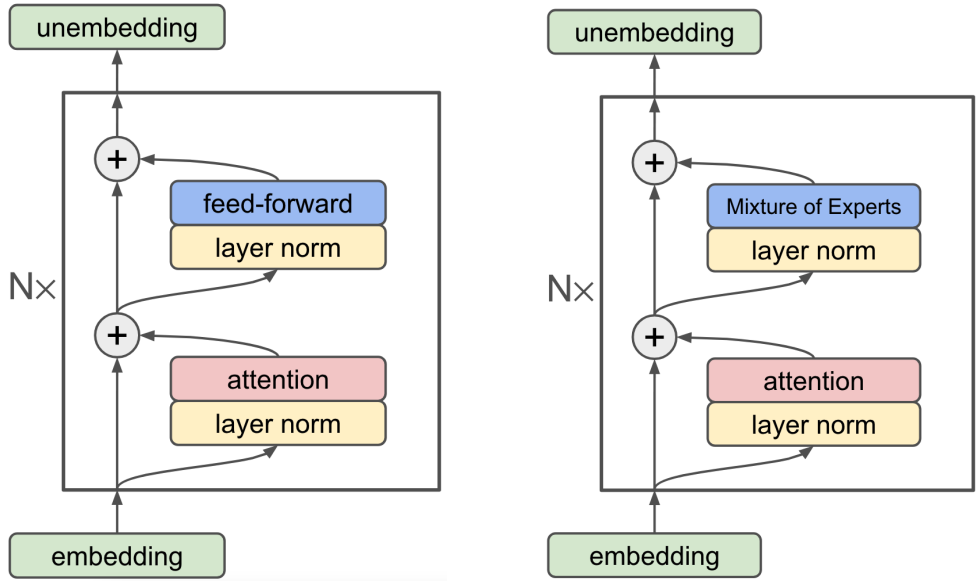


Figure 1.1: **(Left)** Diagram of the decoder-only Transformer architecture. The attention layer is causal attention, also known as self-attention. The Transformer block consists of attention and feed-forward layers. When we say that Transformer has N layers, we mean it has N attention layers interleaved with N feed-forward layers. **(Right)** Diagram of the decoder-only Transformer with Mixture of Experts (see Section 1.6.2) replacing all feed-forward layers. In some models, only every other feed-forward layer is replaced.

Consequently, replacements of the feed-forward layer, like Mixture of Experts, are the primary focus of this work.

1.3 The Computational Cost of Transformers

Generally, the cost of training or inference for any Machine Learning model is proportional to the number of floating-point operations (FLOPs) required, as processing power is the most limiting factor on current hardware. While in practice there are also other constraints on the model besides the total number of FLOPs used—such as the number of memory accesses, total memory used, and required transfer throughput between devices—FLOPs is the most significant. Therefore, it is the most often reported metric used to compare architectures and models[3].

In standard Deep Learning models, including the standard Transformer architecture, the number of FLOPs used per token is linearly proportional to the number of parameters in the model, as most parameters are used precisely once per token. A good approximation of the number of FLOPs when using the model for inference is:

$$\text{FLOPs}_{\text{inference}} = 2 \cdot N \cdot D \tag{1.1}$$

with N being the number of model parameters and D being the total number of tokens processed (including both those provided in the prompt and generated). The constant 2

[3]While metrics like wall-clock time or GPU-hours are sometimes used, they depend on the hardware and implementation details. Therefore, they are often unsuitable as comparison metrics.

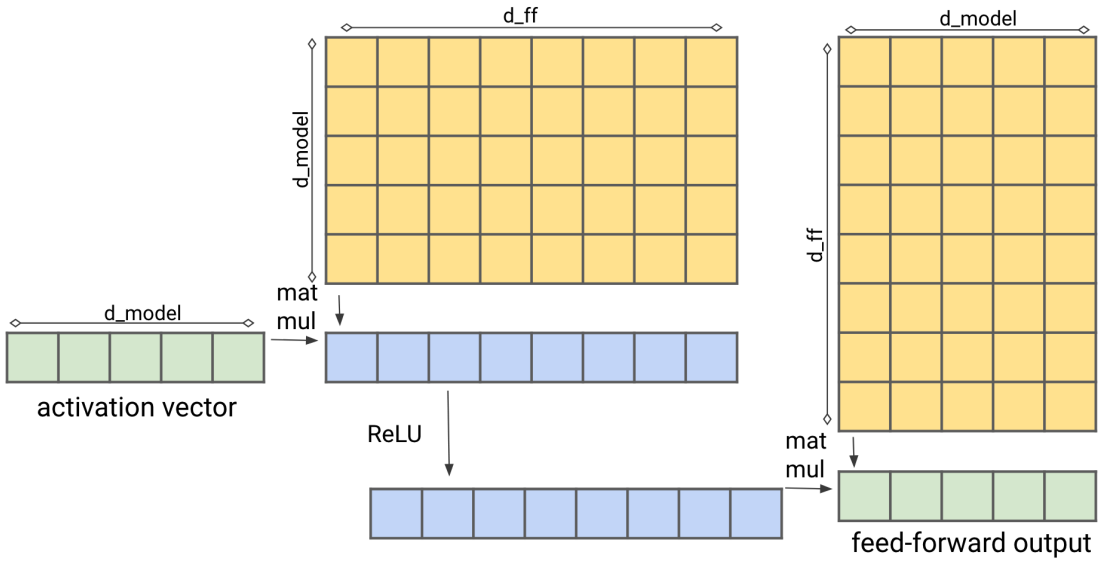


Figure 1.2: Visualization of a simple feed-forward layer with ReLU activation. Weights used for linear projection are colored yellow, while input and output are colored green. Internal activations are colored blue. Biases are not shown for clarity and because they are often skipped in modern Transformers (Chowdhery et al., 2022).

comes from the fact that for each parameter, we perform a single multiplication and a single addition—two operations in total. For training, we can approximate the number of FLOPs using the same equation with the constant changed to 6, accounting for an additional four operations during backpropagation:

$$\text{FLOPs}_{\text{training}} = 6 \cdot N \cdot D \tag{1.2}$$

These approximations, while they do not account for the FLOPs used in the attention mechanism (which depend on the number of tokens in a sequence) and overestimate the number of FLOPs in the embedding layer, are accurate enough to be used in practice, such as in the seminal work of Kaplan et al. (2020).

1.4 Scaling Laws

Scaling laws are empirically derived equations that relate the loss of a model to variables such as the number of parameters, training samples, or the computational budget. In the case of dense Transformers, scaling laws were first studied by Kaplan et al. (2020), who observed power-law relationships between final model perplexity and model and dataset size. This work was extended by Hoffmann et al. (2022), who considered variable cosine cycle lengths and formulated a modified functional form of the scaling equation.

Scaling laws have also been proposed for other architectures and training scenarios. Henighan et al. (2020) studied autoregressive modeling across various modalities, while Ghorbani et al. (2021) considered machine translation. Frantar et al. (2023) explored the impact of pruning on vision and language Transformers, deriving optimal sparsity for a given compute budget. In the context of MoE, Clark et al. (2022) studied the scaling of MoE when altering model size and the number of experts on a fixed dataset, concluding

that routed models are more efficient only up to a certain model size—we revisit this in Chapter 3.

Large Language Models are known to approximately obey the power-law relationship between final loss $\mathcal{L}$, model size N, and number of training tokens D. This relationship is often referred to as the *Chinchilla scaling laws*, described in Hoffmann et al. (2022) as

$$\mathcal{L}(N, D) = c + \frac{a}{N^{\alpha}} + \frac{b}{D^{\beta}}. \tag{1.3}$$

This power-law formula comprises three distinct terms that characterize the intrinsic entropy of data, constraints of the model, and limitations in the training data. The term c represents the minimum possible error intrinsic to the data. The remaining two terms are suboptimality terms that address the limitations in function representation due to the size of the model and in data signified by the number of tokens. With infinite data and model size, the loss is reduced to c in the limit.

The exact parameters in scaling laws are usually fitted to the experimental data, typically with tens or even hundreds of models trained with different values of N and D. See, for example, Table 3.1 in Chapter 3.

1.5 Compute-Optimal Large Language Models

In Section 1.3, we showed the dependency of training FLOPs on variables N and D. In Section 1.4, we described empirical laws detailing the dependency of model loss on these same variables. Taken together, we can now determine the optimal model size N and the number of training tokens D to achieve the best possible model loss $\mathcal{L}$ with a limited computation budget of F FLOPs. This can be done by solving the following optimization problem:

$$\begin{aligned} \underset{N,D}{\text{minimize}} \quad & \mathcal{L}(N, D) \\ \text{subject to} \quad & \text{FLOPs}(N, D) = F. \end{aligned}$$

Models with N and D set to values solving the above optimization problem are referred to in the literature as "compute-optimal." Conversely, models trained longer than this optimal point are called "overtrained," while models trained for shorter durations are called "undertrained." Research such as Kaplan et al. (2020) and Hoffmann et al. (2022) combine empirically fitted scaling laws equations with the FLOPs cost of each model to extrapolate the compute-optimal settings of Transformers far beyond their experimental data. In Chapter 3, we analyze compute-optimal settings of Mixture of Experts models.

When examining current LLMs, it is evident that all these models are overtrained according to scaling laws. This is not a mistake. It is worth noting that classical scaling laws do not consider inference cost when determining the compute-optimal ratio of model size to dataset size. This issue could be mitigated by estimating the expected number of inference tokens during the model's lifetime and adding the cost of this inference to the number of FLOPs. Applying this adjustment always leads to the recommendation of smaller models but trained for longer durations. We argue, then, that the term "overtrained" is a misnomer when referring to most modern LLMs, as they may be, in fact, on the compute-optimality frontier when inference is considered. However, these terms are already ingrained in the literature.

The equations above show the necessity of increasing the model size to train better models. They also demonstrate the linear relation between the number of model parameters and the FLOP count per token processed. At first glance, this seems unavoidable

— when designing the neural network, it seems intuitive to use each parameter exactly once. However, this is just an unnecessary implicit assumption of the design. This work examines methods of using the parameters conditionally, only on some, but not all, possible inputs. This reduces the number of FLOPs per parameter to values possibly much lower than one[4].

1.6 Conditional Computation

1.6.1 Intuition: Neurons as Storage of Information

One can interpret the parameters in a neural network as storage for knowledge. After all, when inferring from a neural network model, the only information about the dataset distribution, including the desired output, is stored in the model parameters. Hence, information from the training set must be processed and stored in the neural network for use during inference. Unfortunately, the model has a limited capacity for storing information.[5] The theoretical limit on the information capacity is the number of bits used to store a particular parameter—usually 32 bits, sometimes 16 or 64. However, in practice, models can be quantized to just a few bits per parameter without much performance loss (Prato et al., 2019). This puts a lower upper bound on the amount of meaningful information stored in the parameter.

Without Conditional Computation, all neural network parameters—and hence all information gathered during the training—are used for every token processed. Intuitively, this seems wasteful—not all knowledge is relevant for every example, and all this processing is extremely costly. The key idea behind Conditional Computation is to decide which parts of the neural network will be relevant for a particular example and skip processing the irrelevant parts.

1.6.2 Mixture of Experts

The most common approach for Conditional Computation is the Mixture of Experts (MoE). It offers an attractive alternative to standard feed-forward layers by drastically increasing the number of parameters without increasing computational cost. MoE was first introduced by Jacobs et al. (1991) as an ensemble-like neural network comprised of separate sub-networks called experts. The core idea is to have multiple *experts*, each specializing in a different part of the input space. Each expert can perform a prediction independently of others, while a router network (also called a gating network or a controller) selects a soft assignment of experts for each input. This generally does not offer computational savings.

Today, mentioning Mixture of Experts most likely refers to a design introduced into Deep Learning by Eigen et al. (2014) and into Natural Language Processing by Shazeer et al. (2017). Here, MoE was integrated as a block into a neural network and operated on the internal representation of the model. This allowed seamless integration with recurrent layers or attention blocks, which used the same set of parameters for each token. Such a

[4]The assumption of using each parameter exactly once can also be rejected in the opposite direction, with a design where each parameter may be used more than once, like the Universal Transformer (Dehghani et al., 2018). However, this approach has not found much practical use.

[5]In classical Machine Learning, the limited capacity of the model was interpreted as necessary to prevent overfitting. However, with the extreme dataset sizes used in Large Language Models, overfitting does not seem to be a practical concern in the traditional sense of harming out-of-distribution performance, apart from overfitting to some evaluation benchmarks.

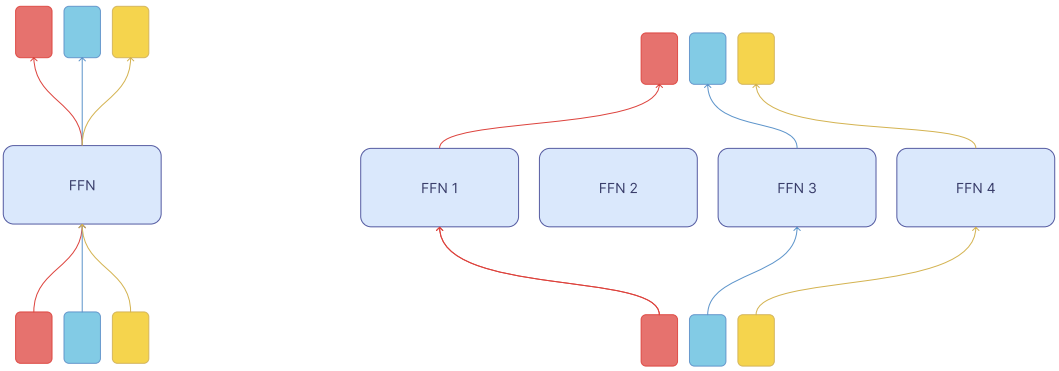


Figure 1.3: **(Left)** Diagram of a standard feed-forward layer featured in the Transformer architecture: each token is processed with the same MLP, independent of other tokens. **(Right)** Diagram of a Mixture of Experts (with Token Choice), where each token chooses the most relevant experts. Note that in the case of an imbalance in the token-expert assignment, some experts may not receive any tokens, while other experts may receive more than average. Compare with Figure 1.4.

design also allows for more fine-grained specialization of experts because the model can choose the most relevant expert at each layer for each token. This provides much-needed flexibility compared to choosing a single expert for the entire process, as was done in prior work. An even more fine-grained approach to MoE is presented in Chapter 3.

Mixture of Experts in the context of Transformers was proposed by Shazeer et al. (2018) and Lepikhin et al. (2020), functioning as a feed-forward layer replacement. See Figure 1.1 for comparison with the standard Transformer. Since the feed-forward layer contains the majority of model parameters and FLOPs, replacing it with a Mixture of Experts is reasonable. While MoE is complex, each expert's design is identical to the feed-forward layer (see Section 1.2 and Figure 1.2). Importantly, an MoE Transformer with a single expert functions the same as a standard Transformer. Hence, individual experts in MoE typically follow all design decisions of the feed-forward layer, particularly the choice of activation function, size of the hidden representation, and inclusion or exclusion of biases. The size of each expert is typically set to mirror the original dimensions of the layer, with the hidden expert dimension d_{expert} equal to d_{ff} (Fedus et al., 2022; Zhou et al., 2022, 2023; Jiang et al., 2024). This keeps the amount of FLOPs per token roughly equivalent to the standard Transformer, assuming a single expert is chosen. For the rest of this section, we assume an MoE layer has N_{experts} experts $\{E_i\}_{i=1}^{N_{\text{experts}}}$, each being a trainable feed-forward network with the same number of parameters.

As of 2024, the Switch Transformer (Fedus et al., 2022) may be considered the most standard design of Mixture of Experts in terms of general structure, routing algorithm, and load balancing. Importantly, it uses common Token Choice routing. With Token Choice, for each token embedding x, we calculate scores $h(x) = Wx \in \mathbb{R}^{N_{\text{experts}}}$, where W is a trainable linear projection. These scores are normalized using softmax:

$$p_i(x) = \frac{\exp\left(h(x)_i\right)}{\sum_{i=1}^{N_{\text{experts}}} \exp\left(h(x)_i\right)}.$$

Prior to Switch, top-k routing, selecting the $k > 1$ most suitable experts for each token, was deemed necessary. However, Switch successfully simplifies previous MoE approaches

by setting $k = 1$ and changing the output of the MoE layer for x to:

$$y = p_I(x)E_I(x),$$

where $I = \text{argmax}_i\, p_i(x)$.

During batched execution, e.g., in training, each batch contains N tokens. If the assignment of tokens to experts is imperfect, i.e., some expert E_f is selected by more than N/N_{experts} tokens in the current batch, the excess tokens are usually dropped and not updated. To encourage an even distribution of tokens to experts, a load-balancing loss, as described by Fedus et al. (2022), is added to the training objective.

There are more techniques to prevent token dropping. An easy but costly solution is to increase the capacity factor for experts. Specifically, with a capacity factor c, each expert may process up to cN/N_{experts}. While higher values of c mitigate token dropping, this additional capacity is usually unused and wasted. Therefore, capacity factors larger than 1 are rarely used during training, as the additional computational cost is generally not worth it. A different approach is to use dropless Mixture of Experts, introduced by Gale et al. (2023), which provides efficient implementation for experts with variable capacity. Lewis et al. (2021) used a linear assignment algorithm to post-process token-expert mappings and ensure even expert selections.

Finally, to prevent token dropping, Expert Choice routing, introduced by Zhou et al. (2022), may be used, which employs a different approach. Instead of the router assigning a single expert to each token, it can allocate a constant number of tokens, usually $N_{\text{tokens}}/N_{\text{experts}}$ per expert. See Figure 1.4 for visualization. This routing may yield better results as it can dynamically assign variable amounts of computation to each token, potentially using more computation on harder tokens and less on easier ones. However, it is more challenging to use in many settings, including during inference in the decoder part of the Transformer.

In more recent works, a fully differentiable soft router was introduced in concurrent works of Chapter 4 for autoregressive language modeling, and by Puigcerver et al. (2023) for image processing.

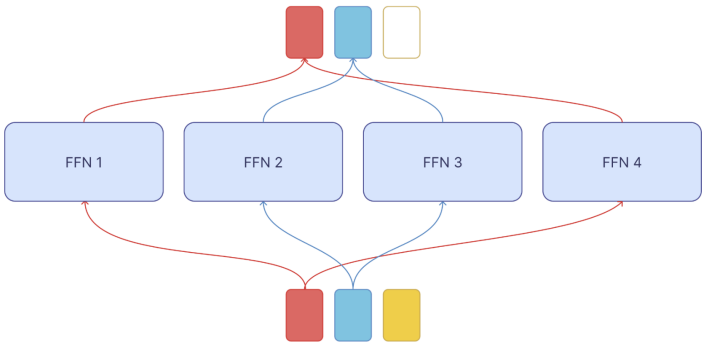


Figure 1.4: Diagram of Mixture of Experts layer with Expert Choice routing. Note that in the case of an imbalance in the token-expert assignment, some tokens may not be processed by any experts, while others may be processed by more than a single expert. Compare with Figure 1.3 (Right).

1.6.3 Other Methods of Conditional Computation

Multiple techniques of Conditional Computation, different from the Mixture of Experts, have been developed. However, MoE remains the most commonly used in the largest models. Furthermore, many techniques not explicitly called Mixture of Experts may be interpreted as special cases of an MoE layer.

Early Exit, such as Xin et al. (2020) and Schuster et al. (2022), allows the model to skip a number of layers at the end, enabling an adjustable amount of computation spent on a sequence or token. While it allows for skipping attention layers as well as feed-forward layers, this same property makes them generally more difficult to train and use.

Similarly, **Mixture-of-Depths** (Raposo et al., 2024) dynamically skips individual layers (including attention layers) during processing, not necessarily at the end. These methods could be compared to MoE layers with two different experts: the first being the entire Transformer block, and the second being a no-op that doesn't change the token representation.

CoLT5 (Ainslie et al., 2023) introduces two variants of each layer: a light variant executed for every token, and a heavy variant executed for a fraction of input tokens.

DeepSeekMoE (Dai et al., 2024), in addition to the standard MoE layer, contains a "shared expert" that executes unconditionally for all tokens.

Mixture of Attention Heads (Zhang et al., 2022) is an adaptation of MoE to the attention layer, where each attention head is a separate expert.

1.6.4 Methods Related to Conditional Computation

Model pruning (Li et al., 2020b; Brix et al., 2020) reduces matrix sizes by eliminating unnecessary weights during or after training. However, gains in computational complexity for sparse matrices often do not translate to inference speedups on actual hardware (Gale et al., 2019). Structured pruning-based approaches (Zhou et al., 2021; Li et al., 2020a; Wang et al., 2020) address this issue by constraining sparsity patterns to be hardware-friendly. For example, only removing whole rows or columns from matrices (Nvidia, 2020).

Mixture of Experts can be interpreted as dynamic structured pruning, as the irrelevant knowledge of the model is pruned dynamically for individual examples. This has clear advantages in terms of model quality for the same compute budget. Typically, the sparsity ratio (the ratio of unused/pruned parameters to used ones) is significantly higher in MoE models. Furthermore, Conditional Computation can be used to accelerate not only inference but also training. However, pruned models generally have a much lower total number of parameters, making them popular on edge devices like consumer smartphones.

Model quantization (Shen et al., 2020; Sun et al., 2019; Prato et al., 2019) uses fewer bits for the weights, essentially compressing each parameter. Conditional Computation is generally complementary to quantization. Although used together, they may introduce greater instabilities into a model. Fortunately, for inference, where quantization is usually applied, the instabilities are rarely a problem.

Model distillation (Sanh et al., 2019) works by training a small student model based on the output of a larger teacher model—distilling its knowledge. This usually works better than training a small model from scratch, as the signal to the model is denser with a distribution over labels instead of a discrete label. This technique is used to speed up inference from pretrained large models. While both MoE and distillation

improve inference performance, model distillation does not speed up training. Conditional Computation is generally complementary to model distillation, with MoE being possible in both the teacher and student models.

1.7 Reproducibility

The code for methods described in Chapter 2 (paper **P1**) is available in our public repository at `https://github.com/google/trax`.

The code for methods described in Chapters 3, 4, and 6 (papers **P2**, **P3**, **P5**) is available in our public repository at `https://github.com/llm-random/llm-random`.

1.8 List of Publications

My thesis comprises of five papers, from **P1** to **P5**:

P1: Sparse is Enough in Scaling Transformers

The work is detailed in Chapter 2.

Authors: **S. Jaszczur**, A. Chowdhery, A. Mohiuddin, Ł. Kaiser, W. Gajewski, H. Michalewski, J. Kanerva

Published at **Conference on Neural Information Processing Systems (NeurIPS) 2021**, CORE Rank: A*, MNiSW: 200

I estimate my contribution to be at 40%.

As the sole first author of this work, my contributions cover the development of the idea, the majority of design, engineering, and experimental and theoretical analysis of methods.

The main contributions of this work are:

1. introducing novel Sparse Feed-Forward layer, enabling conditional computation without degrading the model quality,
2. introducing novel Sparse QKV, a design alternative to linear projections to be used as a replacement in the Attention block,
3. combining the two methods above with a few other techniques to create a Terraformer, and showing the potential of scaling this architecture.

P2: Scaling Laws for Fine-Grained Mixture of Experts

The work is detailed in Chapter 3.

Authors: J. Krajewski*, J. Ludziejewski*, K. Adamczewski, M. Pióro, M. Krutul, S. Antoniak, K. Ciebiera, K. Król, T. Odrzygóźdź, P. Sankowski, M. Cygan, **S. Jaszczur***

Published at **International Conference on Machine Learning (ICML) 2024**, CORE Rank: A*, MNiSW: 200

I estimate my contribution to be at 23%.

As the equal-contribution and the last author of this work, I designed and started the project, supervised it from start to finish, and directly contributed in terms of experiments and analysis.

The main contributions of this work are:

1. introducing a new hyperparameter into MoE explicitly—granularity—allowing increased efficiency of MoE when set correctly,
2. deriving new scaling laws for MoE models, incorporating training duration, the number of hyperparameters, and granularity, enabling computation of optimal hyperparameters for MoE models.
3. demonstrating that, contrary to previous research on MoE scaling laws, MoE models can always outperform traditional Transformers at any computing budget.

P2: Mixture of Tokens: Efficient LLMs through Cross-Example Aggregation

The work is detailed in Chapter 4.

Authors: S. Antoniak*, M. Krutul*, M. Pióro, J. Krajewski, J. Ludziejewski, K. Ciebiera, K. Król, T. Odrzygóźdź, M. Cygan, **S. Jaszczur***

Accepted at **Conference on Neural Information Processing Systems (NeurIPS) 2024**, CORE Rank: A*, MNiSW: 200

I estimate my contribution to be at 25%.

As the equal-contribution and the last author of this work, my contributions cover the idea of a continuous Mixture of Experts, the design of the architecture, and close supervision of the project with further direct and major contributions in terms of implementation, experiments, analysis, and paper writing.

The main contributions of this work are:

1. introducing the novel Mixture of Tokens, a continuous Mixture of Experts architecture that mixes tokens from different examples for joint processing,
2. benchmarking and analysis of MoT, showing the speedup achieved over a dense baseline and the stability over MoE,
3. introducing transition tuning, allowing a pretrained MoT model to be tuned for a sparse MoE inference if desired.

P3: Structured Packing in LLM Training Improves Long Context Utilization

The work is detailed in Chapter 5.

Authors: K. Staniszewski, S. Tworkowski, **S. Jaszczur**, Y. Zhao, H. Michalewski, Ł. Kuciński, P. Miłoś

I estimate my contribution to be at 15%.

As the third author of this work, I helped design the method, later contributing to the code, experiments, analysis, and paper writing.

The main contributions of this work are:

1. introducing the novel SPLiCe method for creating training examples by using retrieval to collate mutually relevant documents into a single training context,
2. fine-tuning OpenLLaMA 3Bv2, OpenLLaMA 7Bv2, and CodeLlama 13B using SPLiCe, showing improved downstream performance on long-context tasks,
3. providing a thorough analysis and ablations of the design choices behind SPLiCe, such as the number and order of retrieved documents.

P4: MoE-Mamba: Efficient Selective State Space Models with Mixture of Experts

The work is detailed in Chapter 6.

Authors: M. Pióro, K. Ciebiera, K. Król, J. Ludziejewski, M. Krutul, J. Krajewski, S. Antoniak, P. Miłoś, M. Cygan, **S. Jaszczur**

I estimate my contribution to be at 15%.

As the last author of this work, I supervised the project from start to finish, including setting the research direction, co-authoring designs, and leading experiments and analysis.

The main contributions of this work are:

1. introducing MoE-Mamba, a model combining Mamba with a Mixture of Experts layer, enabling efficiency gains of both SSMs and MoE while requiring shorter training than vanilla Mamba,
2. demonstrating the robustness of MoE-Mamba improvements across varying model sizes, design choices, and number of experts,
3. providing a thorough analysis and ablations of different alternative methods of integrating Mixture of Experts into the Mamba block.

Supervision and Team-Building

It is worth noting that in three out of these five publications (Mixture of Tokens, MoE-Mamba, and Scaling Laws for MoE), not only did I make significant direct contributions to the work, but the work in general was conducted by the team I initiated and mentored. Apart from my advisor and me, team members had little to no prior research experience in Large Language Models and Conditional Computation, often with no research experience in general. As the team and its members are now an active LLM research group, I consider the creation of this team an additional contribution to my PhD.

Chapter 2

Sparse is Enough in Scaling Transformers

"It is not enough to be busy; so are the ants.
The question is: What are we busy about?"
— Henry David Thoreau

2.1 Introduction

With the growing popularity and size of Transformers, it is increasingly valuable to make them efficient at inference. To address this problem, in this chapter, we study sparse variants for all layers in the Transformer and propose *Scaling Transformers*, a family of next-generation Transformer models that use sparse layers to scale efficiently and perform unbatched decoding much faster than the standard Transformer as we scale up the model size. Surprisingly, the sparse layers are enough to obtain the same perplexity as the standard Transformer with the same number of parameters. We also integrate with prior sparsity approaches to attention and enable fast inference on long sequences, even with limited memory.

Scaling Transformers use a separate *sparse mechanism for the query, key, value, and output layers* (QKV layers for short) and combine it with *sparse feed-forward blocks* to create a fully sparse Transformer architecture.

For QKV layers, our introduced *Scaling Transformers* use only $2d_{\text{model}}\sqrt{d_{\text{model}}} = 2d_{\text{model}}^{1.5}$ parameters (compared to d_{model}^2 in the baseline, fully dense Transformer—see Section 1.3 for details). For the feed-forward layer, our architecture uses Conditional Computation to reduce FLOPs required by nearly two orders of magnitude, while keeping the parameters for storage of information. These changes result in performance as good as the baseline Transformer with the same number of parameters and complexity: $8\,d_{\text{model}}^{1.5} + 4\,d_{\text{model}}^{1.5} + 4\,d_{\text{model}}^{1.5}$.

We were surprised that the fully sparse *Scaling Transformers* are indeed enough to match the results of the baseline Transformer on the large C4 dataset (Raffel et al., 2020) (Figure 2.1). The improvement in complexity holds not just asymptotically but also yields over 2.6x speedup in wall-clock decoding time for a model with 800M parameters and 20x speedup for a model with 17B parameters, as shown in Table 2.1.

To verify that Scaling Transformers can be used with other Transformer improvements on real tasks, we create *Terraformer* — a Transformer model that uses reversible layers for memory efficiency and sparse attention to handle long sequences. We pre-train

	Params	Dec. time	Dec. time per block
baseline Transf.	800M	0.160s	5.9ms
+ Sparse FF	-	0.093s	3.1ms
+ Sparse QKV	-	0.152s	6.2ms
+ Sparse FF+QKV	-	0.061s	1.9ms
Speedup		2.62x	3.05x
baseline Transf.	17B	3.690s	0.581s
+Sparse FF	-	1.595s	0.259s
+Sparse QKV	-	3.154s	0.554s
+Sparse FF+QKV	-	0.183s	0.014s
Speedup		20.0x	42.5x

Table 2.1: Decoding speed (in seconds) of a single token. For the Transformer model (equivalent to T5 large with approximately 800M parameters), Scaling Transformers with proposed sparsity mechanisms (FF+QKV) achieve up to a 2x speedup in decoding compared to the baseline dense model and a 20x speedup for the 17B parameter model.

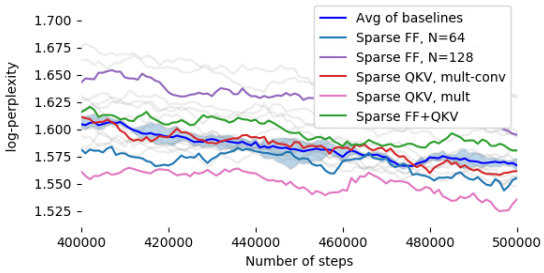


Figure 2.1: The log-perplexity of Scaling Transformers (equivalent to T5 large with approximately 800M parameters) on the C4 dataset, with proposed sparsity mechanisms (FF, QKV, FF+QKV), is similar to the baseline dense model. Other models used in this chapter are shown in gray lines.

Terraformer on the C4 dataset and fine-tune it on the challenging task of summarizing arXiv articles. Terraformer yields results competitive with the state-of-the-art at the time of publication (2021), which at the time was BigBird-Pegasus (Table 2.7)[1].

2.2 Related Work

In this section, we cover work related to this chapter specifically.

Model Compression. Our method differs from pruning approaches as it relies on dynamic sparsity, wherein the feed-forward layer loads only a subset of weights in the layer for each token. Our approach complements model quantization studies (Shen et al., 2020; Sun et al., 2019; Prato et al., 2019), which use fewer bits for the weights.

Sparse attention. Sparse attention-based approaches have made the attention layer more efficient, especially for long sequences, by incorporating additional combinatorial mechanisms, as in Tay et al. (2020a), or by selecting a subset of tokens the layer attends to (Roy et al., 2020; Choromanski et al., 2020; Kitaev et al., 2020; Sukhbaatar et al., 2019; Kaiser & Bengio, 2018; Child et al., 2019) or other approaches (He et al., 2018). Our method is complementary to these approaches for sparse attention and reuses the advances on SOTA therein. Inference speedups in the attention layers also use bottleneck layers (Sun et al., 2020) or grouped convolutions (Iandola et al., 2020). Our work extends beyond the idea of grouped convolutions approach because each attention head is limited to using only a fixed part of the embedding, while our design can permute the embeddings to improve model quality; see Section 2.3.2 for details.

Tensor decomposition. The approaches discussed above significantly improve Transformer speed and handling of long sequences. However, none of them addresses the

[1]In 2024, fully general models like GPT-4o or Claude 3 Opus surpassed the performance of any 2021 state-of-the-art by an extremely wide margin. Metrics like ROUGE scores cannot even meaningfully differentiate between state-of-the-art models of 2024.

fundamental scaling issue: even if we distill into a smaller model, quantize it, and prune a percentage of the weights, the complexity still grows quadratically with the dimension of the model, d_{model}. The final approach, which does tackle this scaling issue, is called *tensor decomposition* in Gupta & Agrawal (2020). Unfortunately, as the authors there note, the approach is most effective in dealing with large input and output embedding matrices and tends to produce lower performance than unstructured models when used inside the decoder.

Conditional Computation. In contrast to earlier Mixture-of-Experts designs, we train a full weight matrix and then only activate specific parts of it for each input token during decoding; see Section 2.3.1. While this limits speedups to inference and not training, it results in superior performance at evaluation time. This is achieved thanks to the granular design of the *Scaling Transformers*. In Chapter 3, we explore a later design of a fine-grained Mixture of Experts, which can be interpreted as an interpolation between *Scaling Transformers* and traditional MoE, combining the advantages of both.

2.3 Sparse Is Enough

We study how to sparsify every part of the Transformer model—otherwise, the non-sparse parts dominate decoding time and become a bottleneck. This means we need sparse equivalents for the feed-forward blocks, the dense Q, K, V, and output layers in attention, and the final dense layer before the softmax and loss.

We run an experiment using an 800M model with 24 layers of encoder and decoder, $d_{\text{model}} = 1024$, 16 attention heads, attention-sparsity = 16, and ff-sparsity = 64. We scale up this model to approximately 17B parameters, with $d_{\text{model}} = 9216$, achieving even a 20× speedup in decoding compared to the baseline dense model. This 17B parameter model has 6 layers of encoder and decoder, 96 attention heads, attention-sparsity = 64, and ff-sparsity = 256.

2.3.1 Sparse Feed-Forward Layer

In a baseline Transformer, decoding speed is dominated by the execution cost of the feed-forward block. Recall that this block consists of two fully connected (dense) layers with a ReLU nonlinearity in between. The dimensionality of activation vectors between these 2 layers is usually denoted by d_{ff} and is often 4 or 8 times larger than the dimensionality of the activations in other places (d_{model}).

We make use of the structure of the feed-forward block to sparsify it. One main observation is that the ReLU in the middle creates a lot of zeros[2]. We impose a fixed structure on this middle activation vector: only one float in every block of N will be allowed to be non-zero. Prior techniques prune weights or blocks from weight matrices and can be referred to as static sparsity. Our proposed technique will train a full weight matrix but only activate specific parts of it for each input token during decoding. We call this dynamic sparsity, because the model dynamically selects only a fraction of its parameters, and the selection is independent for each token.

[2]GeLU is another non-linearity often used in the Transformer feed-forward block. Table 1 in (Narang et al., 2021) shows the same final loss for ReLU and GeLU on the C4 dataset, so here, for simplicity, we focus on ReLU.

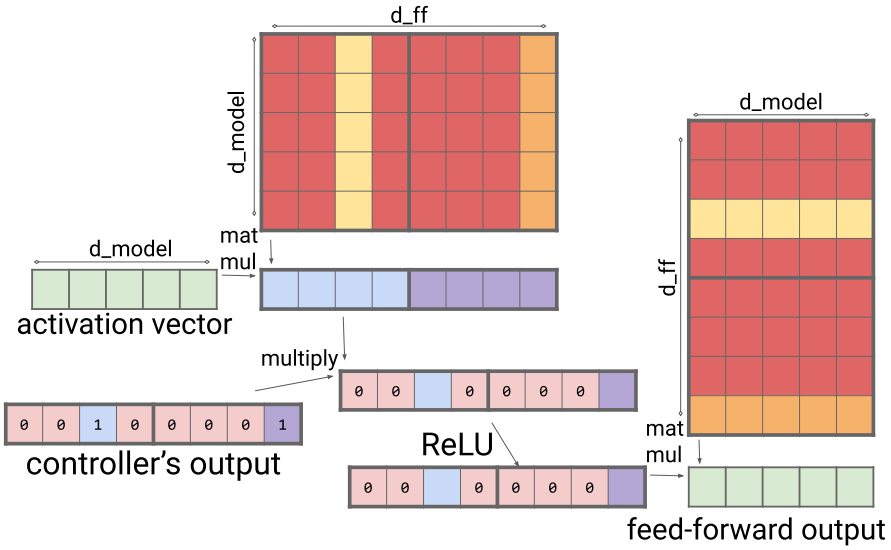


Figure 2.2: Sparse Feed-Forward Layer only activates 1 in N rows/columns of each block to reduce decoding time. Here, only two rows/columns in blocks of size 4 are loaded, while the weights in dark red are not loaded from memory during inference.

We train a router (sometimes called a controller) to determine which activation in each block can be non-zero; the rest will be set to zero. This can be represented as

$$Y_{\text{sparse}} = \max(0, xW_1 + b_1) \odot \text{Router}(x)$$
$$\text{SparseFFN}(x) = Y_{\text{sparse}} W_2 + b_2$$

where $\odot$ is element-wise multiplication. Note that each activation in Y_{sparse} corresponds to a single column in W_1 and a single row in W_2. Therefore, if we compute $\text{Router}(x)$ output first, we don't have to use any columns in W_1 or any rows in W_2 that correspond to an activation set to zero by the router. This allows for much faster decoding, as we have to process only 1 in N columns in W_1 and rows in W_2 (see Figure 2.2(a)).

To design the router to be computationally inexpensive, we project the input using a low-rank bottleneck dense layer. Figure 2.3(b) illustrates the router which produces the output as follows

$$\text{Router}(x) = \arg\max(\text{Reshape}(xC_1C_2, (-1, N)))$$

where $C_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{lowrank}}}$ and $C_2 \in \mathbb{R}^{d_{\text{lowrank}} \times d_{\text{ff}}}$, with d_{lowrank} usually set to (d_{model}/N).

During inference the router uses a discrete argmax function, but during training the model uses a softmax to calculate and sample from a distribution. The model learns to select which row/column will be non-zero using the Gumbel-Softmax trick for discretization. To determine the active row/column in each block, we reparameterize sampling from a Bernoulli distribution by using the Gumbel-Softmax trick (Maziarz et al., 2019). Instead of using the logits in each block to directly sample a binary value, we add independent noise from the Gumbel distribution to each of the logits, and then select the binary value with the highest logit (i.e., argmax) as the sample z. The argmax operation is not differentiable, but it can be approximated by a softmax with annealing temperature. Therefore, on the forward pass, we use the argmax to obtain a binary one-hot vector for

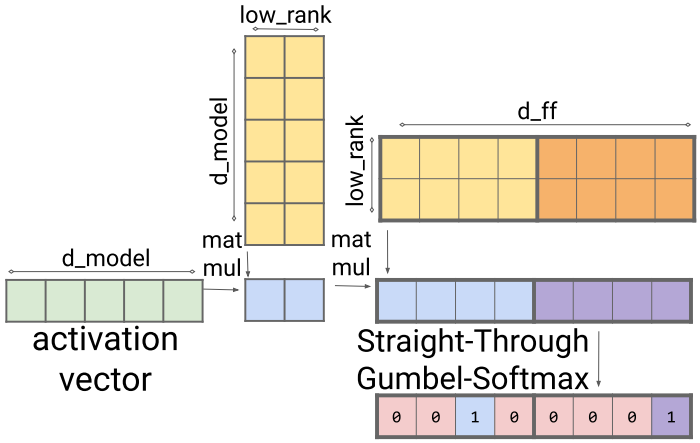


Figure 2.3: Sparse Feed-Forward Router with the output of 2 blocks of size 4 (1 in 4 sparsity).

each block, while on the backward pass, we approximate it with softmax. This approach is known as the Straight-Through Gumbel-Softmax estimator (Jang et al., 2016).

Ablations. We investigate the impact of sparse FF on the model equivalent to T5-large with varying levels of sparsity, with $d_{\text{model}} = 1024$, $d_{\text{ff}} = 4096$, and 16 attention heads. When we set the sparsity level to N (for e.g. $N = 64$) then every block of size N has one non-zero value activated for inference. During training, the router uses the bottleneck layer with $d_{\text{lowrank}} = 64$ and temperature of Gumbel softmax estimator set to 0.1. To improve training stability, the router in the forward pass will use the output of argmax that is a binary one-hot vector for each block with a probability of 30% and otherwise it uses the output of softmax. Table 2.2 and Figure 2.4 show the perplexity and the decoding time of this model with varying levels of sparsity in feed-forward layer. As the level of sparsity increases from 0 to 128, we observe a significant decrease in the decoding time, while the neg-log-perplexity of the model with $N = 64$ sparsity is comparable to the baseline.

We also checked the performance of the feed-forward block with Mixture-of-Experts Shazeer et al. (2017)-style sparsity. As expected, this technique achieved decoding time comparable to sparse FF – 0.11s instead of 0.09s – but with its lack of granularity it achieved log-perplexity of 1.64, worse than both our method and the dense baseline.

2.3.2 Sparse QKV Layer

The decoding speed for a model with Sparse Feed-Forward blocks is dominated next by the query, key, value and output computation—the dense layers in attention, which we jointly call a QKV layer. Each of these dense layers has d^2_{model} parameters and computation cost. Unfortunately, QKV layers don't have ReLUs, so the method used above to sparsify feed-forward blocks is not viable here.

To make QKV layers sparse, we subdivide the dimensionality of the layer, d_{model}, into S modules of size $M = d_{\text{model}}/S$, similar to splitting an activation vector into multiple heads. These modules can be processed with a convolutional layer with fewer weights and faster computation. However, with naïve design each module (and corresponding attention head) could access only a small part of a given token embedding. To alleviate

	Dec. time
baseline	0.160s
Sparse FF 64	0.093s
Sparse FF 128	0.089s

Table 2.2: Decoding time of a single token decreases with increasing level of sparsity in the FF layer.

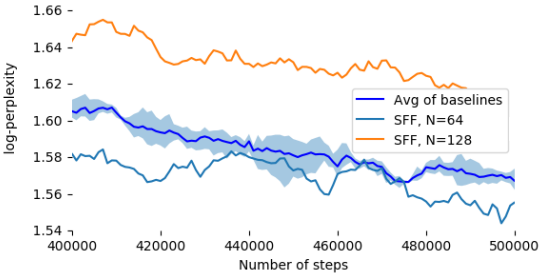


Figure 2.4: Log-perplexity of Scaling Transformers with Sparse Feed-Forward layer is very similar to dense baseline for sparsity level $N = 64$ but degrades slightly for N=128.

that, we develop a multiplicative layer that can represent an arbitrary permutation and has fewer parameters and lower computation time than a dense layer. This multiplicative layer is inserted right before the convolutional layer, letting each head access any part of the embedding (see Figure 2.5(a)). This solution yields well-performing models that also decode fast.

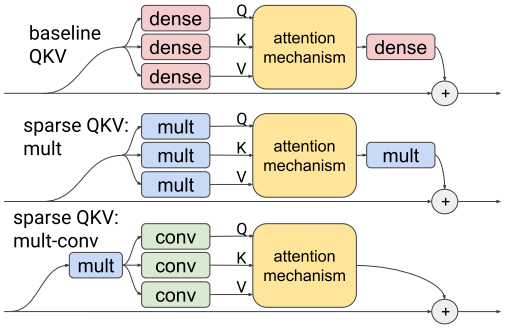


Figure 2.5: Sparse QKV layer replaces Q, K, and V dense layers by composing multiplicative and convolutional layers and reducing the number of parameters and decoding time.

Multiplicative dense layer. Our new multiplicative dense layer can represent an arbitrary permutation and has $d_{\text{model}}^2/S + d_{\text{model}}S$ parameters, dependent on the sparsity hyperparameter S. It processes an input vector $\mathrm{x} \in \mathbb{R}^{d_{\text{model}}}$ by splitting it into S "modules" of size $M = d_{\text{model}}/S$. It produces output $\mathrm{y} \in \mathbb{R}^{S \times M}$ as follows

$$\mathrm{y}_{s,m} = \sum_i \mathrm{x}_i D_{i,s} E_{i,m}$$

where the two weight matrices are $D \in \mathbb{R}^{d_{\text{model}} \times S}$, and $E \in \mathbb{R}^{d_{\text{model}} \times M}$ (see Figure 2.6(b)). This layer executes significantly faster during inference because of the decreased number of parameters which need to be loaded from memory. Unless stated otherwise, we use $S = 16$.

The multiplicative layer is designed primarily to represent any permutation, so that each attention head can access information from any part of the embedding. We first verify that the multiplicative layer can indeed represent an arbitrary permutation.

Theorem. *With Multiplicative layer defined as*

$$y_{s,m} = \sum_i x_i D_{i,s} E_{i,m}$$

For any bijective function $f : \{1 \cdots d_{model}\} \Rightarrow \{1 \cdots S\} \times \{1 \cdots M\}$ *there exists a pair of weights of multiplicative layer D, E such that* $x_i = y_{s,m}$ *for* $\{s, m\} = f(i)$.

Proof. Let's take a function f, and define functions $s, m : s(i), m(i) = f(i)$. We construct weights $D_{i,s'} = (1 \text{ if } s' = s(i) \text{ otherwise } 0)$ and $E_{i,m'} = (1 \text{ if } m' = m(i) \text{ otherwise } 0)$. With those constraints we can derive, from the definition of multiplicative layer:

$$\mathrm{y}_{s',m'} = \sum_i (\mathrm{x}_i \text{ if } D_{i,s'} = 1 \wedge E_{i,m'} = 1 \text{ otherwise } 0)$$

$$\mathrm{y}_{s',m'} = \sum_i (\mathrm{x}_i \text{ if } s' = s(i) \wedge m' = m(i) \text{ otherwise } 0)$$

$$\mathrm{y}_{s',m'} = \sum_i (\mathrm{x}_i \text{ if } f(i) = s', m' \text{ otherwise } 0)$$

Because function f is injective we can use its inversion.

$$\mathrm{y}_{s',m'} = \sum_i (\mathrm{x}_i \text{ if } i = f^{-1}(s', m') \text{ otherwise } 0)$$

$$\mathrm{y}_{s',m'} = \mathrm{x}_{f^{-1}(s',m')}$$

$$\mathrm{y}_{f(i)} = \mathrm{x}_i$$

□

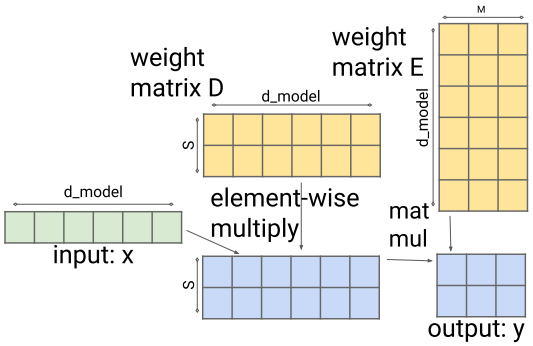


Figure 2.6: Multiplicative layer can represent an arbitrary permutation, but has fewer parameters and reduced computation time compared to a dense layer.

Convolutional layer. The output of the multiplicative layer is a tensor of type/shape $\in \mathbb{R}^{\text{batch} \times \text{length} \times S \times M}$. We process this tensor with a two-dimensional convolutional layer, treating the length dimension and number of modules S like height and width of an image. This layer uses M filters and a kernel size of $F \times F$ so that each filter looks at F modules ('S' axis) of the last F tokens ('length' axis). Replacing the standard dense layer with such a convolution reduces the parameter count and computation time of the QKV

layer. At the same time, by convolving over the 'length' axis, the model can incorporate more context into this computation (Li et al., 2019).

The output of this layer has the same shape as the input. The optimal value of S is less than $\sqrt{d_{\text{model}}}$. Empirically we set F to 3, S equal to the number of heads in the attention mechanism and M to be the dimensionality of a single attention head. In this case, we can feed the output of the convolution directly to the attention mechanism without reshaping the output. This convolutional layer has fewer parameters ($9M^2 + M = F^2(d_{\text{model}}/S)^2 + (d_{\text{model}}/S)$), and lower computational complexity ($O(d^2_{\text{model}}/S)$). Unless stated otherwise, we use $S = 16$ and $F = 3$.

Combining multiplicative and convolutional layers. There are four dense layers to replace in the original attention mechanism: Q, K, V, and output. As shown in Figure 2.5(b), we replace Q, K, and V dense layers by composing multiplicative and convolutional layers, but with a multiplicative layer shared across all three: $Q = \text{conv}_Q(\text{mult}(x))$, $K = \text{conv}_K(\text{mult}(x))$, $V = \text{conv}_V(\text{mult}(x))$. We remove the output dense layer. Note that the combined multiplicative-convolutional variant has the output dense layer removed, while the other variants have it replaced with their respective sparse layers. Including this output layer negatively impacts decoding time. We can set the parameter S to around $\sqrt{d_{model}}$, getting the number of layer parameters to scale proportionally to $d^{1.5}_{model}$ compared to d^2_{model} of standard QKV layer.

Interpretation of QKV layer. Note that when parameter S in convolutional layer is equal to the number of heads in the attention mechanism, which is the case in our experiments, then each of the S modules corresponds to a single attention head. Therefore, the model uses the convolution to process each head using the same linear projection. Without the multiplicative layer this projection would operate on a predetermined part of the embedding layer for each head. However, by adding it the model can perform arbitrary permutation of dimensions, so each head can have access to arbitrary subset of embedding dimensions, not a predetermined subset of them. This fact helps with keeping the expressibility of resulting QKV layer despite the reduced number of parameters.

Ablations. We investigate the impact of sparse QKV layers on the model equivalent to T5-large in Figure 2.7. We increase the value of d_{ff} from 4096 to 6144 to preserve the number of parameters (see the next subsection for details). The decoding time with sparse QKV layer variants is similar to the baseline because it is dominated by the dense feed-forward layer.

Combined feed-forward and QKV sparsity. Sparse QKV layers lower the total number of model parameters. To keep the model size matched to the baseline, we increase d_{ff} to keep the number of parameters similar across all models we compare. For the T5-Large equivalent model, we increase d_{ff} from 4096 to 6144. With increased d_{ff}, decoding time in the feed-forward layer increases and thus, Sparse QKV layers alone do not speed up the model. However, when we combine Sparse QKV layers with sparse FF layers, we get a 3.05x speedup in decoding time of each decoding block with comparable perplexity (see Table 2.1 and Figure 2.1). While the baseline these is a vanilla Transformer, the decoding speed is almost the same for a Reformer model as well.

Table 2.4 shows the accuracy of fine-tuning for downstream tasks from the GLUE dataset. Note that the model with sparseFF+QKV and the baseline have similar accuracy.

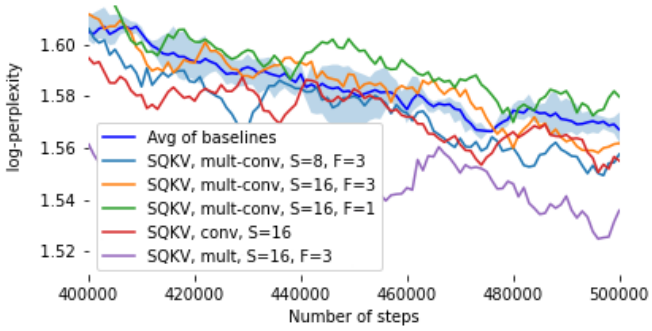


Figure 2.7: Log-perplexity of Scaling Transformers with Sparse QKV with different sparsity levels (S) and kernel sizes (F) is very similar to dense baseline within variance while multi-layer even improves perplexity.

Sparse loss layer. A final dense layer maps the model embedding into vocabulary size to compute the loss. We can sparsify this part of the model by replacing the dense layer with a multiplicative layer designed for Sparse QKV layer. This change speeds up decoding time but may degrade perplexity. Table 2.3 and Figure 2.10 shows that increasing the sparsity of the loss layer degrades the perplexity slightly while speeding up the decoding time. While the absolute difference in decoding time is negligible when not using Sparse FF and Sparse QKV at the same time, it provides considerable performance when used jointly; see Table 2.8.

Sparse loss	Dec. time
baseline	0.160 s
S=2	0.158 s
S=4	0.149 s
S=8	0.148 s

Table 2.3: Decoding times by varying the number of modules S in sparse loss layer.

2.4 Terraformer - Sparsity for Long Sequences

The above gains from sparsifying the dense layers are encouraging, but we omitted one fundamental issue. When applied to longer sequences, the gains would effectively be lost, as the decoding time will be dominated by attention operations. Luckily, a number of

	RTE	MRPC	SST-2	QNLI	MNLI-m	QQP
Baseline Transformer (dense)	70.1 ± 1.1	83.6 ± 0.72	92.6 ± 0.85	88.6 ± 0.5	78.5 ± 0.41	85.2 ± 0.6
Scaling Transformer (Sparse FF+QKV)	68.4	81.2	91.6	90.1	82.9	89.9
Terraformer (Sparse FF+QKV)	66.1	84.6	92.3	88.3	79.1	85.5

Table 2.4: Accuracy of Scaling Transformer model and Terraformer model with sparse QKV+FF is comparable to the baseline Transformer within variance. The results are obtained by fine-tuning on selected downstream tasks from the GLUE dataset (validation split).

methods have been proposed to solve this problem for Transformers, see Tay et al. (2020b) for a survey. We focus on the LSH (Locality-Sensitive Hashing) attention from Reformer (Kitaev et al., 2020) and show how to integrate this sparse attention mechanism, as well as recurrent blocks, into a Scaling Transformer, yielding a *Terraformer*.

2.4.1 Architecture for Long Sequences

While integrating sparse attention layers into a Scaling Transformer, we notice that the architecture of the Transformer decoder block is suboptimal and can be redesigned to make a better use of these layers. In particular, separating decoder self-attention and encoder-decoder attention is not necessary any more from the perspective of efficiency. We therefore remove the encoder-decoder attention, but just concatenate the encoder representations before the decoder tokens. Doing this alone isn't enough though, since we took away one attention mechanism (encoder-decoder attention). We remedy this by having two attention mechanisms before the feed-forward block. This simple architecture is as fast as the baseline Transformer while giving better results.

Putting this together, if v_{enc} are the encoder activations and v_{dec} are the decoder embeddings, the input to the decoder block x is their concatenation on the length axis, $\text{LengthConcat}(v_{enc}, v_{dec})$. Each decoder block can be represented as:

$$\begin{aligned} y_1 &= x + \text{Dropout}(\text{Attention}(\text{LayerNorm}(x))) \\ y_2 &= y_1 + \text{Dropout}(\text{Attention}(\text{LayerNorm}(y_1))) \\ y &= y_2 + \text{FFN}(y_2) \end{aligned}$$

where y becomes the input to the next decoder layer. See Figure 2.8 for the diagram of the architecture.

2.4.2 Reversibility for Memory Efficiency

To enable training Terraformer with large batches, and to fine-tune even large models on single machines, we apply ideas from the Reformer (Kitaev et al., 2020), in particular, reversible layers for the encoder and decoder blocks.

The original Reformer decoder block contained feed-forward and attention layers in a 1-1 ratio. In the Terraformer architecture, as described above, there are two attention layers in the decoder block, so there are three swaps in the reversible layers in the decoder block (see Figure 2.9). In our experiments, this significantly improved performance.

Another issue with reversibility is that it is only formally correct for continuous functions. We find that this is not just a formal issue, but an important problem in practice. To make reversible layers train well with sparsity, we need to store the discrete decisions—i.e., the integers saying which rows to select—and use them for reversing. Recalculating these decisions on the backwards pass leads to worse results.

2.4.3 Recurrence for Generalization

In addition to incorporating sparse attention and reversibility, we also add recurrence to the feed-forward block of Terraformer. Recurrent layers allow information to propagate in time, even in a single decoder block. It is challenging though to use them without decreasing model speed, esp. in training. For that reason, we use simple recurrent units (Lei et al., 2017) which parallelize well during training.

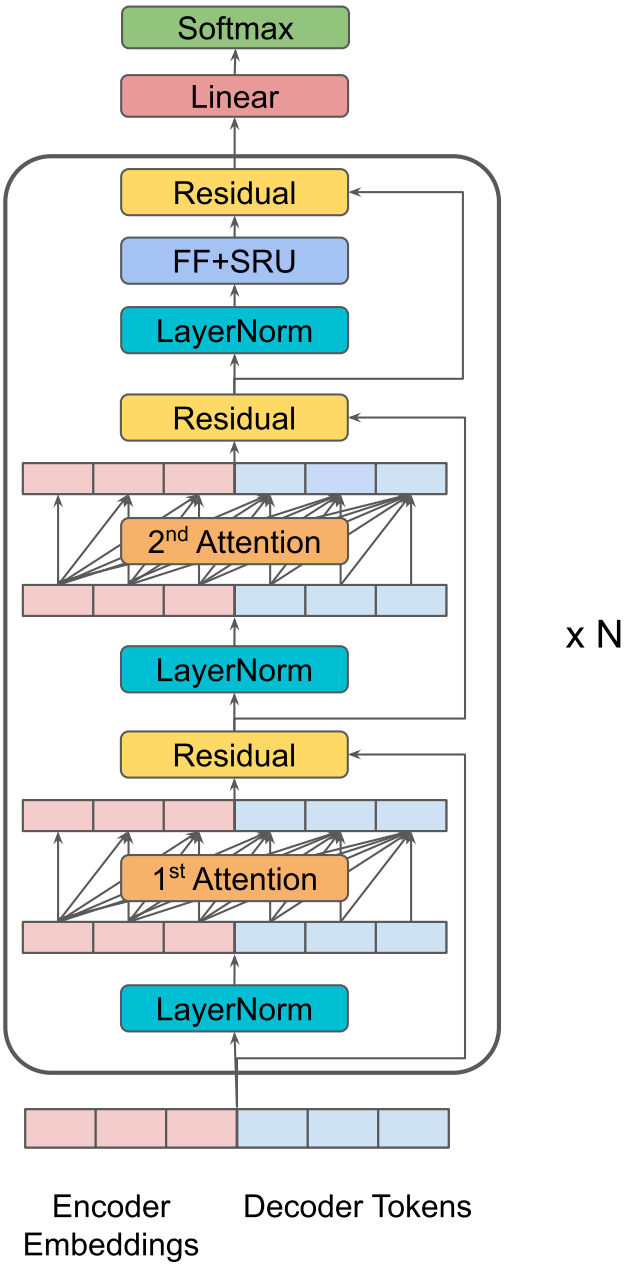


Figure 2.8: Terraformer Architecture uses two attention mechanisms before the feed-forward block in each decoder block.

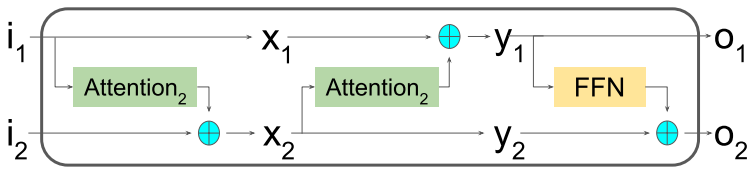


Figure 2.9: Reversible decoder block in Terraformer.

SRUs contain dense layers, so their use could negate the benefits of sparsity elsewhere. We tried a few methods to alleviate that, but it turns out that simply reducing the dimensionality of the SRUs works. So we first project from d_{model} to a small dimension (32 in our experiments), then apply the SRU, and then project back to d_{model} and add the result to the feed-forward block. This low-rank recurrence is in our experiments sufficient to transfer enough information through time for the network to generalize.

Since the effects of SRUs on C4 are minimal (as the training and evaluation data are very similar), we use synthetic tasks to investigate out-of-distribution generalization. We train the models on long addition and on the task of copying a decimal digit. We train on inputs with at most 128 digits and evaluate on inputs lengths from 256 to 300, so over 2x longer. As can be seen in the table below, the baseline Transformer does not generalize well, while Terraformer manages to get a large portion correctly, even if it is not perfect like the Neural GPU (Kaiser & Sutskever, 2015).

Model	copy	copy (seq)	add	add (seq)
Transformer	79.8%	0%	36.4%	0%
Terraformer	99.9%	93.9%	86.9%	32.4%

Table 2.5: Comparison of out-of-distribution generalization for Terraformer and Transformer on two toy tasks, long addition and copying on decimal numbers. Under (seq) we report the number of fully correct sequences generated as answers.

2.4.4 Pretraining Terraformer on C4 Dataset

We pretrained Terraformer in the same way as all other baselines reported in this chapter with the same number of parameters (800M), the same dimensions as mentioned before, and loss sparsity of 4 to get the fastest model. The only difference is 4 times larger batch size as, thanks to reversibility, Terraformer can be trained with larger batches.

Table 2.6 shows the perplexity and decoding speed of the Terraformer model in comparison to the baseline Transformer model and the sparse Transformer model from the previous section. All models have the same number of parameters (800M) and the same dimensions as mentioned before. We used loss sparsity 4 for Terraformer to get the fastest model, so in Table 2.6 we compare it to a sparse Transformer with the same sparsity in loss layer.

2.4.5 Downstream Evaluations of Terraformer

We designed Terraformer so that the benefits from sparsity would not be lost on long sequences, nor on downstream finetuning tasks. To test this, we chose the task of summarizing scientific papers using the dataset of scientific papers from arXiv (Cohan et al., 2018). In this task, the input is a whole paper—a long sequence—and the model is asked to output its abstract. Several recent papers studied this dataset and tasks and it has been shown (Zhang et al., 2020; Zaheer et al., 2020) that pretraining on C4 yields significant improvements on this task. We find that Terraformer is competitive with the above baselines, even though we mask single words (we do not use the Pegasus sentence loss) and decode the answers in a greedy way (no beam search). Note that ROUGE scores are computed using open-source scorer[3] with the metrics described in its

[3] `https://pypi.org/project/rouge-score/`

	steps	batch size	Log perpl.	Dec. time
baseline Transf.	500k	4	1.57	0.160s
sparse Transf.	500k	4	1.61	0.061s
Terraf.	125k	16	1.66	0.086s
Terraf.	150k	16	1.63	0.086s
Terraf.	175k	16	1.59	0.086s

Table 2.6: Terraformer (800M) trained with 4x larger batch size achieves log-perplexity similar to baseline dense Transformer and Scaling Transformers with sparse FF+QKV and sparse loss. Terraformer trained with larger batch size does not match the perplexity of the baseline at $\frac{1}{4}$th number of steps, but catches up at around $\frac{1}{3}$rd—we believe this may be due to the fact that we used training hyper-parameters optimized for the baselines. Decoding of a single token is 1.92x faster than baseline.

Model	R-1	R-2	R-LSum	R-LSent
Terraformer	45.40	17.86	41.21	26.33
DANCER RUM	42.70	16.54	38.44	—
BIGBIRD-RoBERTa	41.22	16.43	36.96	—
Pegasus Large (C4)	44.21	16.95	38.83	25.67
DANCER PEGASUS	45.01	17.6	40.56	—
BIGBIRD-Pegasus	46.63	19.02	41.77	—

Table 2.7: Terraformer is competitive with strong baselines (Zhang et al., 2020; Zaheer et al., 2020; Gidiotis & Tsoumakas, 2020) on the ArXiv summarization task, without using the Pegasus loss and without beam search. On R-1, R-2 and R-LSum, Terraformer outperforms all previous models except for BigBird-Pegasus.

documentation[4]. We also observe certain confusion between ROUGE-L metrics reported. As noted in the open-source scorer, there are two versions of ROUGEL-Sentence-Level (R-LSent) and ROUGEL-Summary-Level (R-LSum). For clarity, we report both of these metrics. Furthermore we only report the F1 measure of any ROUGE metric. We include a few examples of the generated abstracts in the Appendix A.

We also observe that the Terraformer model achieves accuracy similar to the Transformer model in Table 2.4 for selected downstream tasks on GLUE dataset.

Table 2.8 shows the speedup in decoding with sparse layers when we scale up Terraformer to 17B parameters. Note that sparsifying all the layers gives us 37x speedup in decoding.

[4] `https://github.com/google-research/google-research/tree/master/rouge`

Terraformer	Dec. time	Speedup
dense	3.651s	1x
Sparse FF	1.595s	2.29x
SparseFF+QKV	0.183s	19.98x
SparseFF+QKV+loss	0.097s	**37.64x**

Table 2.8: Decoding speed of a single token for Terraformer with 17B parameters is 37x faster than a dense baseline model, requiring less than 100ms/token for inference. Here attention-sparsity = 64, ff-sparsity = 256, and loss-sparsity = 4.

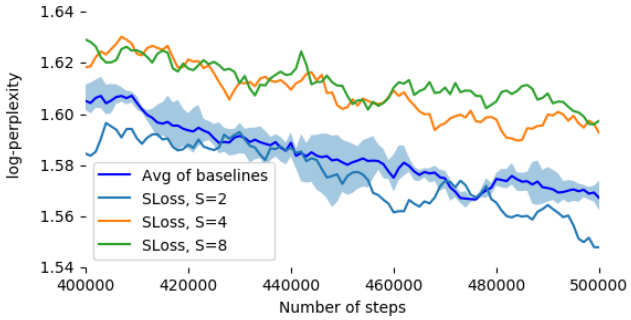


Figure 2.10: Log-perplexity of baselines and Scaling Transformers with just Sparse Loss, and varying number of modules.

2.5 Conclusions

When starting to investigate sparse variants of Transformers, we assumed that there would be a price to pay for sparsity—that a sparse model would always underperform a dense one with the same number of parameters. To our surprise, this is not the case: sparse is enough!

In our experiments with large models on the C4 dataset, the sparse models match the performance of their dense counterparts while being much faster at inference. When scaling the models up, the benefits of sparsity become even larger. This promises to put Transformers back on a sustainable track and make large models more useful. Moreover, there are numerous techniques for making models faster that could greatly benefit Terraformer and other Scaling Transformers. For example, we did not study quantization, and we believe that it can make Scaling Transformers even faster. With proper tuning and further improvements, we believe one could train a Scaling Transformer to match GPT-3 in accuracy but also run inference in a reasonable time on a laptop.

Our results have several limitations. For one, the practical speedups we see are only for inference, not at training time. Moreover, we consider unbatched inference on CPUs, while inference is often run in batched mode on GPUs. Those limitations stem from the extreme granularity of conditional computation used in this chapter, as this approach is unsuitable for modern hardware accelerators like GPUs or TPUs.

However, we lift all of those limitations in Chapter 3 by introducing an additional hyperparameter of granularity, where Sparse FF can be interpreted as an extreme case of maximal possible sparsity, and common MoE variants like Switch Transformer can be interpreted as an extreme case of minimal possible sparsity.

Chapter 3

Scaling Laws for Fine-Grained Mixture of Experts

In medio tutissimus ibis.
"You will be safest in the middle."
— Ovid, *Metamorphoses*

3.1 Introduction

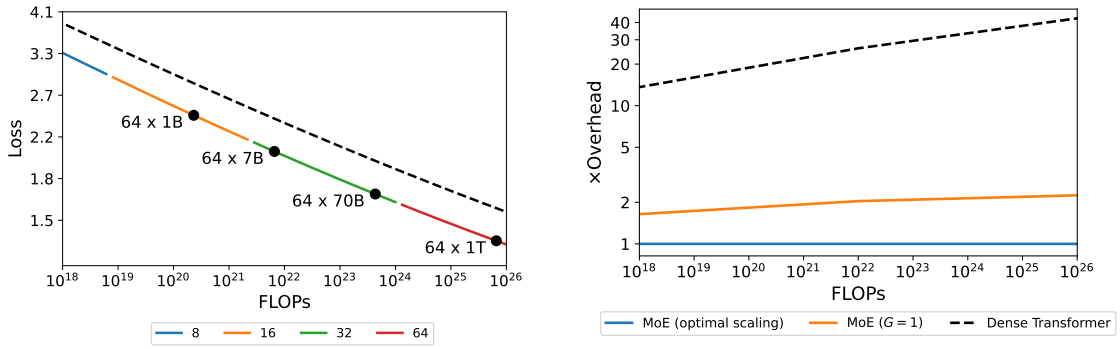
In the context of the increasing budgets for training models, a question arises: Will MoE models continue to be attractive in the future? This is an important issue, as results from other studies (Clark et al., 2022) suggest that traditional dense models may outperform MoE as the size of the models increases.

In this chapter, we argue that previous claims lose their validity when we relax certain implicit assumptions regarding the training process, as presented in previous research (Clark et al., 2022). In particular, we refer to the fixed training duration and the constant size of experts in MoE models.

We also introduce a new hyperparameter, granularity, the modification of which allows for the optimal adjustment of the size of experts. While most previous MoE models could be interpreted as a special case of minimal granularity, the methods presented in Chapter 2 could be interpreted as maximum granularity. Considering possible architectures that are efficient on modern hardware accelerators, we show that optimal granularity lies somewhere in the middle.

We derive the scaling laws for Mixture of Experts, incorporating the following into the equations: the model size, the previously neglected number of training tokens, and the newly introduced concept of granularity. We show that all of them have a meaningful impact on the scaling properties of MoE models.

Using the scaling laws, we show that, with optimal settings, MoE models can always outperform traditional Transformers at any computing budget. This conclusion is contrary to the results from Clark et al. (2022); see Section 3.5.3. Our results suggest that a compute-optimal MoE model trained with a budget of 10^{20} FLOPs will achieve the same quality as a dense Transformer trained with a $20\times$ greater computing budget, with the compute savings rising steadily, exceeding $40\times$ when a budget of 10^{25} FLOPs is surpassed (see Figure 3.1).



(a) **(b)**

Figure 3.1: Mixture of Experts can be *always* considered more efficient than dense Transformers, regardless of the model size. **(a).** Compute optimal scaling curves for granular models and standard Transformers. The dashed line represents a dense Transformer. Colors denote optimal granularity for the given FLOPs training budget. **(b).** Relative number of FLOPs needed to train Transformer and vanilla MoE (MoE with $G = 1$) to achieve the performance of MoE with compute optimal G.

3.2 Related Work

In this section, we cover work related to this chapter specifically.

3.2.1 Mixture of Experts

Concurrently to the publication of the work presented in this chapter, Dai et al. (2024) proposed to modify the MoE layer by segmenting experts into smaller ones and adding shared experts to the architecture. Independently, Liu et al. (2023b) suggested a unified view of sparse feed-forward layers, considering, in particular, varying the size of memory blocks. Both approaches can be interpreted as modifying granularity. However, we offer a comprehensive comparison of the relationship between training hyperparameters and derive principled selection criteria, which these works lack.

3.2.2 Scaling Laws

Scaling laws for non-MoE Transformers and other Deep Learning architectures were covered in detail in Chapter 1.

The most significant previous work on scaling laws of MoE models was performed by Clark et al. (2022). They formulated the final loss for a constant dataset size D of 130B tokens, allowing for variations in the number of experts E, as:

$$\mathcal{L}(N, E) = \left(\frac{10^{d/a}}{N}\right)^{a} \left(\frac{1}{E}\right)^{b+c\log N}. \tag{3.1}$$

However, this equation has a notable limitation as it can be applied only to the original dataset size. Scalability and effectiveness are constrained in this scenario because it is crucial to align the number of training samples with available computational resources

for optimal use. As per Kaplan et al. (2020) and Hoffmann et al. (2022), maintaining a constant dataset size while scaling up the neural network size leads to undertraining, resulting in a model that does not perform to its full potential. See Chapter 1 for a detailed discussion of scaling laws.

For the aforementioned reasons, Clark et al. (2022) concluded that routed models are more efficient only until a certain model size. In this chapter, we challenge that claim by considering a variable, optimal dataset size for both model families (see Section 3.5.3). We also introduce an additional dimension of granularity into the scaling laws.

3.3 Granularity

As described in Section 1.6.2, in the standard setting, the inner dimension of each expert network d_{expert} is equal to the dimension of the feed-forward layer, d_{ff}, of the base model.

In this chapter, we propose an alternative approach where the hidden dimension of the expert is not necessarily set to mirror that of the standard feed-forward layer. Instead, it can be adjusted to a value that is the most effective. This approach allows the configuration of MoE to be articulated in terms of two key hyperparameters: *granularity* (G) and *expansion rate* (R). In the following parts of this chapter, we will also use the term *active* parameters to refer to the non-embedding parameters used to produce output for a single token, excluding routing. The number of active parameters is denoted as N_{act}.

Let d_{expert} be the hidden dimension of a single expert. Granularity is defined as

$$G = \frac{d_{\text{ff}}}{d_{\text{expert}}}.$$

In other words, granularity denotes the multiplication factor for the change in the size of an expert from the original standard model, defined as $G = 1$. In this chapter, we investigate $G > 1$, where experts are smaller than those in the standard layer.

Note that increasing granularity does not affect the number of active parameters. As G increases, the number of experts processing the token grows proportionally to G. That is, for granularity G, a token is routed to G fine-grained experts, keeping the number of active parameters constant. See Figure 3.2 for visualization.

We then define the *expansion rate*, which describes the increase in the number of parameters from a standard Transformer layer to an MoE layer. Given that, N_{MoE} and N_{ff} denote the total number of parameters in an MoE layer (excluding routing) and in the standard feed-forward layer, respectively. The expansion rate R is then defined as

$$R = \frac{N_{\text{MoE}}}{N_{\text{ff}}}.$$

The expansion rate can also be seen as the total number of parameters in an MoE layer compared to active parameters.

The concept of the expansion rate is intricately linked to the number of experts through the idea of granularity. Indeed, the definitions of both granularity and expansion rate extend and refine our understanding of the number of experts, symbolized as N_{expert}.

$$N_{\text{expert}} = G \cdot R \tag{3.2}$$

For non-granular models, where $G = 1$, the expansion rate is equal to the number of experts.

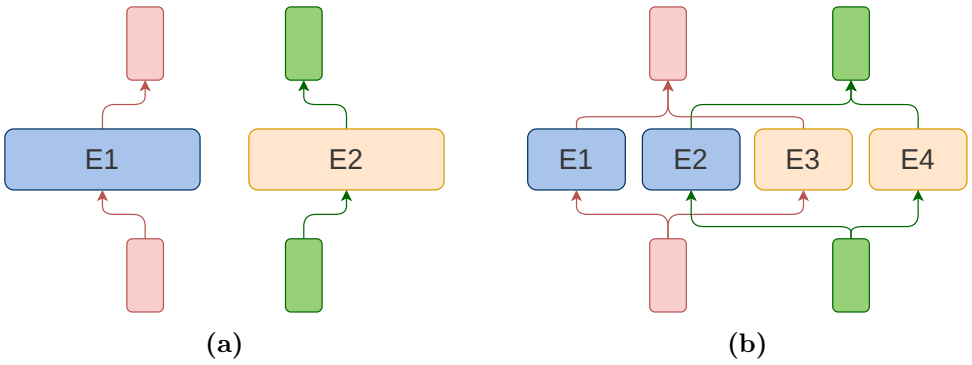


Figure 3.2: **(a)** Standard MoE layer with $G = 1$. **(b)** Corresponding MoE layer with $G = 2$. Each of the original experts is split into two granular ones. The split occurs in the hidden dimension of an expert. Increasing G allows for a more precise mapping between experts and tokens. Since for granularity G, the token is routed to G granular experts, the number of parameters activated per token is the same in both cases.

Intuitively, increasing granularity for a given expansion rate provides the model with greater flexibility in mapping datapoints to experts, potentially improving performance. We incorporate the notion of granularity into our scaling laws in Section 3.4. The discussion about practical trade-offs in adjusting this parameter is provided in Section 3.5.

3.4 Scaling Laws

Introducing granularity changes the architecture of MoE. In this section, we answer a central question arising: whether the granular MoE models follow scaling laws and, if so, how granularity affects them. Thus, we aim to derive a parametric scaling law for predicting the final loss value $\mathcal{L}$ based on granularity G, the total number of non-embedding parameters N, and the number of training tokens D.

We run over 100 experiments on the decoder-only Transformer architecture, with each feed-forward component replaced by an MoE layer. These experiments involve training models with sizes ranging from 129M to 3.7B parameters across different training durations, from 16B to 130B tokens. We consider logarithmically spaced values of granularity between 1 and 16. To constrain the search space, $R = 64$ is fixed, following the recommendations of Clark et al. (2022). In addition, we also run experiments with dense Transformers to compare their performance with MoE. The details of all architectures, the training procedure, and hyperparameter choices are described in detail in Section 3.6.

In the subsequent parts of this chapter, we will use the notation $R \times N_{\text{act}}$ to describe an MoE model with N_{act} active parameters and expansion rate R.

3.4.1 Power Law with Respect to Granularity

We first answer the question of whether granular models follow the scaling laws. In Figure 3.3, we can observe that increasing granularity results in a lower loss. The returns follow approximately an exponential pattern, converging to a positive constant. The empirical relationship shown in Figure 3.3 suggests, for a given N and D, the following

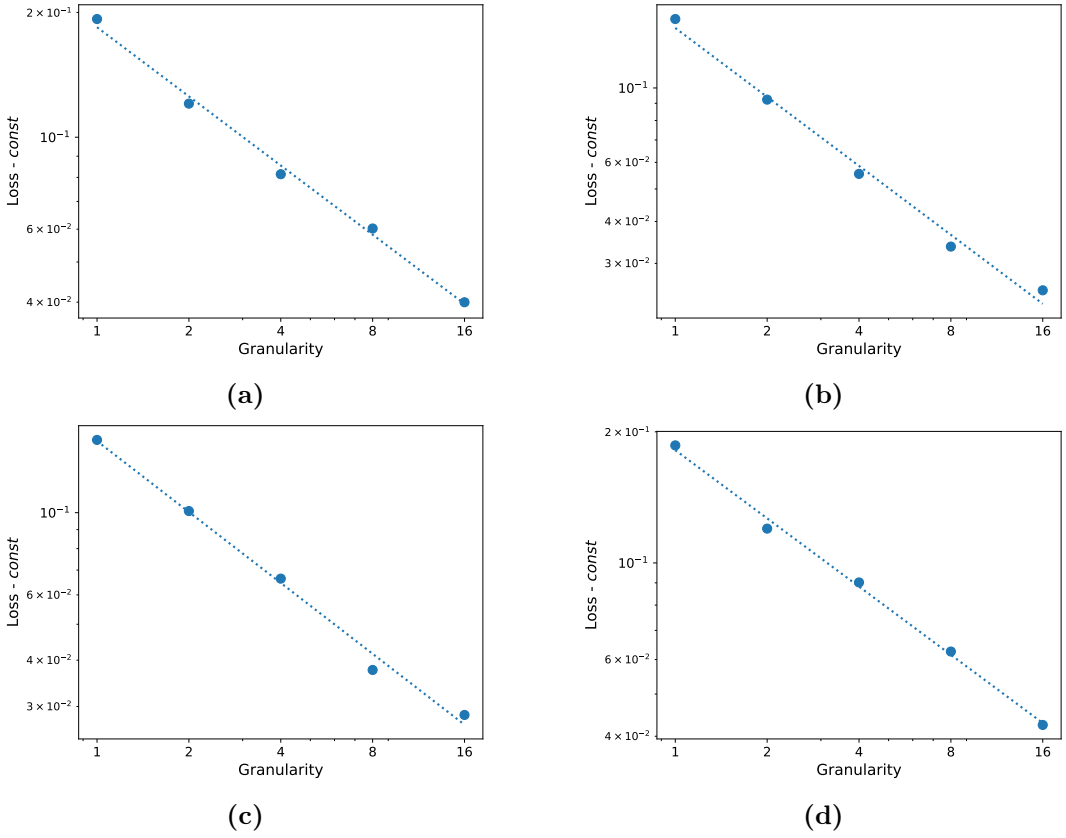


Figure 3.3: We plot the effect of G on $\mathcal{L}_{N,D}(G)$ for constant N and D. Both axes are in log scale. The results suggest a linear relationship between $\log(G)$ and $\log(\mathcal{L} - c)$. The given values are: **(a).** $N = 64 \times 25M$, $D = 16B$, $const = 3.12$ **(b).** $N = 64 \times 49M$, $D = 16B$, $const = 3.02$ **(c).** $N = 64 \times 25M$, $D = 32B$, $const = 3.03$ **(d).** $N = 64 \times 49M$, $D = 32B$, $const = 2.88$

power-law dependence of loss $\mathcal{L}_{D,G}$ on varying granularity G, parametrized by $g_{N,D}$, $\gamma_{N,D}$, and $h_{N,D}$,

$$\mathcal{L}_{N,D}(G) = \frac{g_{N,D}}{G^{\gamma_{N,D}}} + h_{N,D}. \tag{3.3}$$

3.4.2 Scaling the Model and Dataset Size

As outlined in Chapter 1, the Chinchilla scaling law given by Equation 1.3 — for convenience repeated below in Equation 3.4 — consists of three terms that describe inherent data entropy and limitations in function representation and data.

$$\mathcal{L}(N, D) = c + \frac{a}{N^{\alpha}} + \frac{b}{D^{\beta}}. \tag{3.4}$$

This derivation is independent of the architecture. In particular, Equation 3.4 also holds for constant granularity. Empirically, we observe a power-law relationship in N and D analogous to that in dense models (see also Figure 1 in Kaplan et al. (2020)), as depicted in Figure 3.4 for a fixed value of granularity. Furthermore, the validity of this functional form is verified by fitting in Section 3.4.4.

Since we know that separate scaling laws are valid for given granularities, in the general form, the parameters in Equation 3.4 can depend on the model's granularity:

$$\mathcal{L}_G(N, D) = c_G + \frac{a_G}{N^{\alpha_G}} + \frac{b_G}{D^{\beta_G}}. \tag{3.5}$$

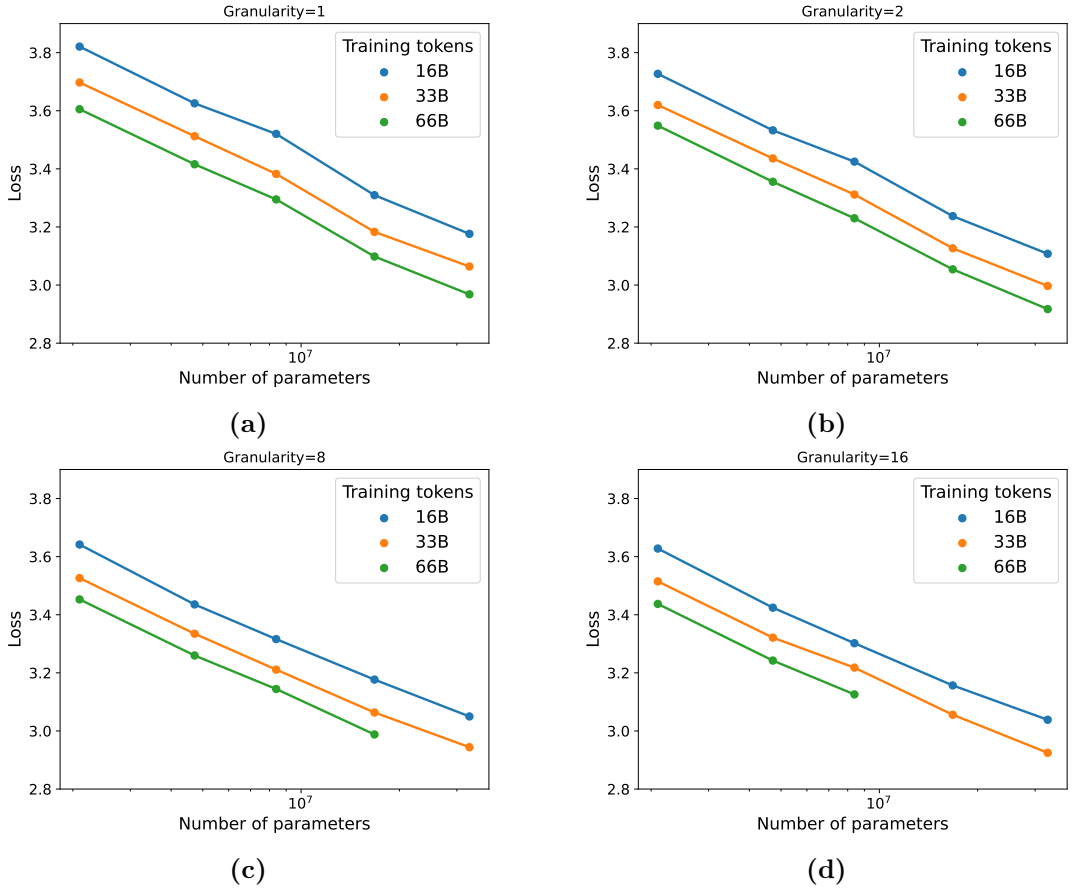


Figure 3.4: The impact of varying the number of parameters, N, on the loss for fixed granularity, G: **(a)** $G = 1$, **(b)** $G = 2$, **(c)** $G = 8$, **(d)** $G = 16$.

3.4.3 The Form of the Joint Scaling Law

Following the observation that models with constant granularity obey Chinchilla scaling laws given by Equation 3.4, the key question is how to incorporate the general notion of granularity G into the joint scaling law.

Moreover, the scaling law formula from Equation 3.5, for constant N and D, must be representable by Equation 3.3. This is because the former is a more general equation, encompassing shared hyperparameters across all N, D, and G. It is anticipated to align with the latter, consisting of distinct power laws, each with specific parameters for different N and D values. Consequently, the objective is to identify a function that fulfills these criteria.

$$\mathcal{L}(N, D, G) = \mathcal{L}_{N,D}(G) = \mathcal{L}_G(N, D) \tag{3.6}$$
$$= \frac{g_{N,D}}{G^{\gamma_{N,D}}} + h_{N,D} = c_G + \frac{a_G}{N^{\alpha_G}} + \frac{b_G}{D^{\beta_G}}$$

In the subsequent sections, we aim to determine which of these parameters remain independent of G and identify their functional form. Additionally, we provide some rationale for the structure of our formula.

Lower bound. Consider the limit of Equation 3.5 as N and D grow to infinity:

$$\lim_{\substack{N\to\infty \\ D\to\infty}} \mathcal{L}(N, D, G) = c_G. \tag{3.7}$$

with the constant term c_G dependent on granularity.

This dependence is contradictory to the fact that the term captures the inherent entropy of the dataset, as defined in Hoffmann et al. (2022), Appendix D.2: *'the minimal loss achievable for next-token prediction on the full distribution P, also known as the "entropy of natural text."'*.

The lower bound of the achievable loss for training larger models on more samples should not depend on the architecture since it is a function of the dataset, not the model. Therefore, the parameter $c_G = c$ remains constant for all granularities.

Granularity and Number of Tokens D. As seen in Figure 3.5, the benefit of training a model on a larger dataset is consistent across different granularity values. This suggests that there is no interaction between D and G. Therefore, we can assume that

$$\frac{b_G}{D^{\beta_G}} = \frac{b}{D^{\beta}}. \tag{3.8}$$

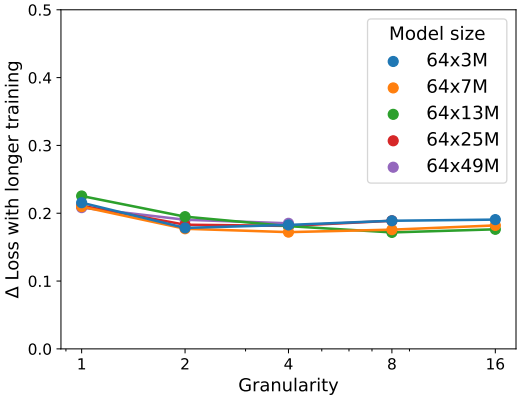


Figure 3.5: Difference in loss between training for $16B$ and $65B$ tokens across all model sizes and granularity values. The model size is reported as the expansion rate and a number of active parameters.

Granularity and Model Size N. We consider α to be a constant describing how the function scales with N. In this chapter, we assume polynomial functional forms that

rule out the potential dependency of α on G, given the form of Equation 3.3. Therefore, the only element dependent on G is a_G:

$$\mathcal{L}(N, D, G) = c + \left(\frac{g}{G^{\gamma}} + a\right) \frac{1}{N^{\alpha}} + \frac{b}{D^{\beta}}. \tag{3.9}$$

Finally, one could consider omitting the constant a in the equation above, and it would still reduce to Equation 3.3 for constant N and D. However, this would imply that a model with infinite granularity and a small number of active parameters can achieve the perfect perplexity of the lower bound. We believe that an MoE sparse model should not surpass the performance of its dense counterpart matched by the total number of parameters, with all of them activated. This indicates that constant a can act as a marginal improvement from granularity.

3.4.4 Fitting the Parametric Scaling Law

Subsequently, we fit parameters in Equation 3.9 to describe the scaling of MoE. For comparison, we also perform fitting for a dense Transformer given by Equation 3.4. Similar to Hoffmann et al. (2022), we use Huber loss (Huber, 1964), with $\delta = 0.1$. The optimization is performed using the BFGS algorithm. We include a weight decay of $5e-4$ to enhance generalization. We start by fitting parameters in Equation 3.9, and then find architecture-dependent coefficients α, β, a, and b in Equation 3.4. The values are presented in Table 3.1. We depict the fit of the equation in Figure 3.6. We generally observe a good fit, with RMSE = 0.015.

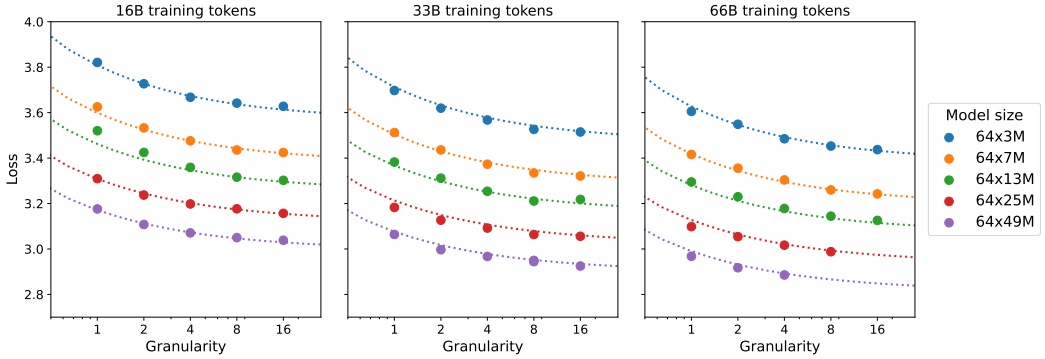


Figure 3.6: The fit of the scaling law compared to experimental results.

Model	a	α	b	β	g	γ	c
MoE	18.1	0.115	30.8	0.147	2.1	0.58	0.47
Dense	16.3	0.126	26.7	0.127	-	-	0.47

Table 3.1: Values of the fitted coefficients.

We validate the stability of the fit by excluding the top 20% of models with the lowest perplexity and finding the coefficients based on the remaining experiments. We observe that the formula remains almost unchanged in this scenario (see Table 3.2). The validation RMSE is 0.019. Results are depicted in Figure 3.7.

Model	a	α	b	β	g	γ	c
MoE	17.6	0.114	26.7	0.140	2.07	0.570	0.472

Table 3.2: Values of the fitted coefficients with 20% of data points with the lowest perplexity excluded for validation.

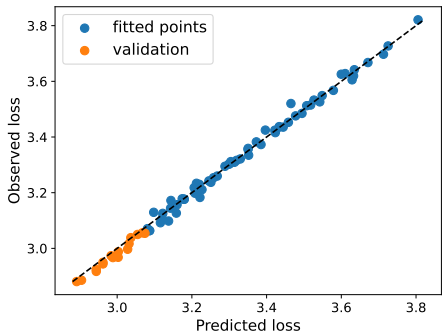


Figure 3.7: Validation of the fit with the exclusion of the top 20% of models with the lowest perplexity.

3.4.5 MoE Scaling Properties

Comparing the part of the formula that approximates underfitting (that is, dependent on training tokens) in MoE ($30.8D^{-0.147}$) and Transformer ($26.7D^{-0.127}$), we infer that MoE models require longer training to perform competitively but scale better after reaching that point. Nonetheless, this moment may still precede the compute-optimal for both models. On the other hand, we can see that the exponent on dense models, $\alpha = -0.126$, scales better with the total number of parameters than the MoE counterpart, $\alpha = -0.115$. This should not be surprising since dense models use all parameters for each token, contrary to MoE, which gains a computational advantage by activating only a subset of parameters. Therefore, a fair comparison of performance must consider the FLOPs used by each model type. In the next section, we find the compute-optimal granularity for a given FLOP budget.

3.5 Optimal Allocation of Computational Budget

The goal of this section is to find the optimal values for N, D, and G given a computational budget F. This can be done by solving the following optimization problem:

$$\begin{aligned} \underset{N,D,G}{\text{minimize}} \quad & \mathcal{L}(N, D, G) \\ \text{subject to} \quad & \text{FLOPs}(N, D, G) = F. \end{aligned}$$

3.5.1 Computational Cost of Granularity

It is important to acknowledge that increasing granularity can lead to some challenges in training the model, such as higher computational and communication costs and a larger memory footprint.

The main component responsible for higher costs is the increase in routing operations due to a larger pool of granular experts. This increase is proportional to the value of G. For standard, non-granular MoE models ($G = 1$), the routing overhead still exists, although it has been considered negligible.

Taking into account the routing operation overhead, the number of used FLOPs F is described by the following formula:

$$F = (12{d_{\text{model}}}^2 c_f + d_{\text{model}} RGc_r) \cdot D \cdot n_{\text{blocks}}, \tag{3.10}$$

given expansion rate R, granularity G, and constants that denote FLOPs per active parameter ratio, respectively, within routing (c_r) and within the rest of the network (c_f). The term $12{d_{\text{model}}}^2$ is the number of active parameters within a Transformer block, while $d_{\text{model}}RG$ is the number of active parameters within a routing network. Following Hoffmann et al. (2022), we calculate the number of FLOPs per active parameter and derive c_f to be 6 and c_r to be 14. We exclude embedding and unembedding from the FLOPs calculations, also following Hoffmann et al. (2022).

Observe that, in contrast to scenarios where routing operations are omitted, the FLOPs calculation that incorporates routing overhead relies on both d_{model} and n_{blocks}. Consequently, an additional condition is required to determine the scaling of d_{model} and n_{blocks} in relation to an increase in N, the number of parameters. It is noted that minor variations in the depth-to-width ratio are not significant (Kaplan et al., 2020). Following this analysis, we opt to adopt the assumption that $d_{\text{model}} = 64n_{\text{blocks}}$.

The total number of parameters in the feed-forward layer, excluding the routing matrix, is $2Rd_{\text{ff}}d_{\text{model}} = 8R{d_{\text{model}}}^2$, and $4{d_{\text{model}}}^2$ in attention (key, query, value, and output projection). This results in the following formula for $N = {d_{\text{model}}}^2 \cdot (8R+4) \cdot n_{\text{blocks}}$.

Modeling the cost of granularity on hardware accelerators. It is important to note that the exact estimation of the training cost of MoE models depends on the training setup, hardware, and implementation. Specifically, increasing G can lead to higher transfer costs, depending on the adopted model of distributed training. Therefore, the precise selection of hyperparameters should be made considering these factors. In this chapter, we model the cost of operations using FLOPs, a common approach in the scaling laws literature (Kaplan et al., 2020; Hoffmann et al., 2022; Frantar et al., 2023). Additionally, we would like to note that in our setup, we observe significant gains for granular models measured as wall-clock time needed to achieve a given perplexity; see Section 3.6.

3.5.2 Compute Optimal Formula

Putting it all together, we need to solve the following optimization problem, given F,

$$\begin{aligned} \underset{N,D,G}{\text{minimize}} \quad & \mathcal{L}(N, D, G) \\ \text{subject to} \quad & F = (12{d_{\text{model}}}^2 c_f + d_{\text{model}} RGc_r) \cdot D \cdot n_{\text{blocks}} \\ & N = d_{\text{model}}^2 \cdot (8R + 4) \cdot n_{\text{layers}}, \\ & d_{\text{model}} = 64 \cdot n_{\text{layers}}. \end{aligned}$$

All these constraints can be reduced to a one-dimensional optimization problem, which is, however, difficult to solve analytically. Therefore, we approximate the solution using Brent's method (Brent, 1971). The results of this optimization for various FLOPs budgets are plotted in Figure 3.1 while the optimal configurations of parameters for

selected model sizes are presented in Table 3.3. To validate the uncertainty of these predictions, we follow Hoffmann et al. (2022) and calculate the 10th and 90th percentiles estimated via bootstrapping data (80% of the data is sampled 100 times). See Table 3.4 for details.

N	D	G	FLOPs	Loss
64 x 100M	4.37B	8	2.95e+18	3.133
64 x 1B	28.94B	16	1.93e+20	2.491
64 x 3B	72.90B	16	1.41e+21	2.245
64 x 7B	137.60B	32	6.46e+21	2.076
64 x 70B	941.07B	32	4.16e+23	1.694
64 x 300B	2.96T	64	5.69e+24	1.503
64 x 1T	7.94T	64	4.97e+25	1.367

Table 3.3: Compute optimal training hyperparameters for MoE models. Optimal N and D follow a similar relation to those in Hoffmann et al. (2022) for active parameters around the range of $1B$ to $10B$. Smaller models require comparatively longer training, while larger models require shorter training. Note that this also considers optimal granularity and its FLOPs cost.

N	D	G
64 x 100M	(2.97B, 5.98B)	(8, 8)
64 x 1B	(21.17B, 40.73B)	(16, 16)
64 x 3B	(50.20B, 105.88B)	(16, 32)
64 x 7B	(101.06B, 205.40B)	(32, 32)
64 x 70B	(638.49B, 1.59T)	(32, 64)
64 x 300B	(1.99T, 5.62T)	(64, 64)
64 x 1T	(5.29T, 16.87T)	(64, 64)

Table 3.4: 10^{th} and 90^{th} percentiles estimated via bootstrapping data.

3.5.3 MoE is Always More Efficient

Contrary to the results from Clark et al. (2022), in Figure 3.1 we observe that Mixture of Experts can always be considered more efficient than dense Transformers, regardless of model size. According to our previous observations in Section 3.4.5, MoE models scale better with optimal training. However, for short training schedules, they may underperform dense models. This means that for a constant training time and increasing model size, there exists a point where both models become very under-trained; in this scenario, dense models surpass MoE. This explains why in Clark et al. (2022), where varying the number of training tokens was not considered, MoE was predicted to underperform for models larger than 1T. However, when all training hyper-parameters N, D, G are properly selected to be compute-optimal for each model, the gap between dense and sparse models only increases as we scale.

3.5.4 Assumption of Fixed Dataset Size

Clark et al. (2022) focuses on a fixed dataset size $D = 130B$ and defines the effective parameter count of any MoE model as the size of a dense model that achieves the same perplexity as a given MoE with a certain number of active parameters. Subsequently, this effective parameter count curve intersects with the active parameter curve, indicating that using MoE will lead to worse performance in the future. Despite this incompatibility, we do not claim their experiments and extrapolations are invalid—it is just the assumption of a constant dataset size. If we use our fitted scaling laws with a fixed number of training tokens D, the resulting *e*ffective parameter count curve (which can be properly defined only for a fixed D) would indeed intersect at some parameter count similarly to Clark et al. (2022).

The crossing point of the *effective parameter count* curve, that is, a number of parameters where dense Transformers surpass MoE models for given number of training tokens D, can be calculated by determining where our Scaling Laws cross, solving the following equation for N, given D: $0.47 + (\frac{2.1}{G^{0.58}} + 18.1)N^{-0.115} + 30.8D^{-0.147} = 0.47 + 16.3N^{-0.126} + 26.7D^{-0.127}$ This solves for:

D	10B	130B	1T
(N) MoE/dense Crossing Point	251B	1.9T	10T

Table 3.5: Crossing point for MoE Effective Parameter Count curve with dense

This crossing point will be placed further away for each increase in D.

Importantly, the lines will never cross if we train each of these models in a compute-optimal manner using the same computational budget (Figure 3.1). Our results show that regular dense models perform better than MoE only when both are severely undertrained, as seen in the extrapolation from Clark et al. (2022), where a comparison is made between $1T$ parameter models trained on $130B$ steps. Moreover, in industry settings, it is common to overtrain LLMs and extremely rare to undertrain them, and our scaling laws indicate even better performance of MoE models in overtrained regimes.

3.6 Architecture and Training Setup

All models considered in this chapter are decoder-only Transformers trained on the C4 dataset (Raffel et al., 2020). We use the GPT-2 tokenizer (Radford et al., 2018). Each batch consists of 0.5M tokens packed into 2048 sequences. Our optimizer is AdamW (Loshchilov & Hutter, 2019), with a weight decay of 0.1. In each training run, we use a maximum learning rate of 2e−4, with linear warmup for 1% of steps and cosine decay to 2e−5. To improve stability, we initialize weights using the truncated normal distribution with reduced scale, as advised in Fedus et al. (2022). The models are trained using mixed precision; we always keep the attention mechanism and router in high precision. We assume the *infinite data* regime, as the number of training tokens in any of the runs is less than the number of tokens in the corpus. We follow Hoffmann et al. (2022) and perform our analysis on the smoothed training loss.

In MoE, we use the Expert Choice routing algorithm, as it guarantees a balanced expert load without tuning additional hyperparameters. To maintain compatibility with autoregressive language modeling, we apply the recipe described in Zhou et al. (2022):

tokens are grouped by position across different sequences. The group size is always set to 256. We match the number of FLOPs for MoE and dense models with the same d_{model} (meaning we activate an average of $8d^2_{\text{model}}$ parameters per token in each MoE layer). In the router, softmax is performed over the expert dimension, while we choose tokens over the token dimension, as this leads to the best performance (as opposed to performing softmax over the token dimension). We put an additional layer normalization before the output of the MoE layer. This gives a small improvement for standard MoE but is crucial for the performance of models with $G > 1$.

#parameters (nonemb)	d_{model}	n_{blocks}	n_{heads}	D (in #tokens)	G
64x3M	256	4	4	16B, 33B, 66B	1, 2, 4, 8, 16
64x7M	384	4	6	16B, 33B, 66B	1, 2, 4, 8, 16
64x13M	512	4	8	16B, 33B, 66B	1, 2, 4, 8, 16
64x13M	512	4	8	130B	1, 2, 4
64x25M	512	8	8	16B, 33B,	1, 2, 4, 8, 16
64x25M	512	8	8	66B	1, 2, 4, 8
64x49M	640	10	10	16B, 33B	1, 2, 4, 8, 16
64x49M	640	10	10	66B	1, 2, 4
64x85M	768	12	12	33B	1, 2, 4

Table 3.6: Architecture and training variants (MoE models).

Table 3.6 and Table 3.7 list the considered architecture and training variants for dense and MoE models, respectively.

To show that FLOPs improvements translate to efficiency gains on modern hardware accelerators, we provide an example of training curves for models with different levels of granularity, measured in terms of wall-clock training time on NVIDIA A100 GPU. We can see that the model with $G = 8$ achieves the best performance in this case. See Figure 3.8.

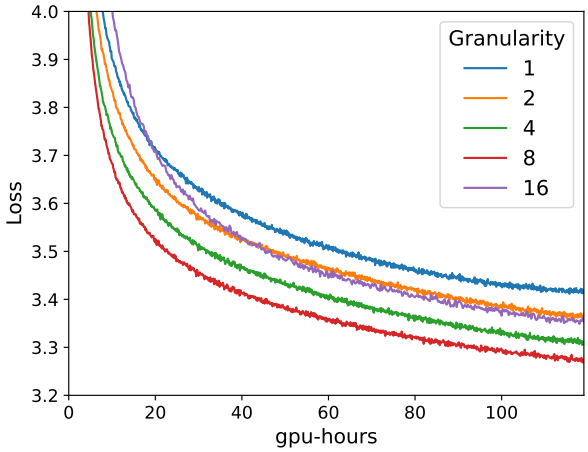


Figure 3.8: Training loss curves for model with $N = 64 \times 7M$, $D = 66B$ tokens.

#parameters (nonemb)	d_{model}	n_{blocks}	n_{heads}	D (in #tokens)
3M	256	4	4	16B, 24B, 33B, 66B
6M	256	8	4	16B, 24B, 33B, 66B
13M	512	4	8	16B, 24B, 33B, 66B
25M	512	8	8	16B, 24B, 33B, 66B
49M	640	10	10	16B, 24B, 33B, 66B
85M	768	12	12	16B, 33B

Table 3.7: Architecture and training variants (dense models).

3.7 Varying Expansion Rate

Due to computational resources constraint, this chapter focuses on $R = 64$, as recommended by Clark et al. (2022). This value of R was also used for the largest models in other works Du et al. (2022); Zhou et al. (2022) and the best-performing configuration in Fedus et al. (2022). Nonetheless, we acknowledge the importance of considering different expansion rates, as different levels of R may be chosen based on factors like the target size of the model in memory. Therefore, in this section, we provide results for $R = 16$ and show that the main findings of this chapter are still valid in such cases. The models considered in this part are listed in Table 3.8.

#parameters (nonemb)	d_{model}	n_{blocks}	n_{heads}	D (in #tokens)	G
16x3M	256	4	4	8B, 16B, 33B	1, 2, 4, 8, 16
16x7M	256	8	4	8B, 16B, 33B	1, 2, 4, 8, 16
16x13M	512	4	8	8B, 16B, 33B	1, 2, 4, 8, 16
16x13M	512	4	8	66B	1, 2, 4
16x25M	512	8	8	8B, 16B, 33B	1, 2, 4, 8, 16
16x49M	640	10	10	8B	1, 2, 4, 8, 16

Table 3.8: Architecture and training variants (MoE models).

We fit Equation 3.9 using the same procedure as described in Section 3.4.4. The results are detailed in Table 3.9.

Model	a	α	b	β	g	γ	c
MoE ($R = 16$)	19.64	0.124	57.07	0.169	1.18	0.986	0.472

Table 3.9: Values of the fitted coefficients.

Using the coefficients and FLOPs calculation formulas, we can derive the compute-optimal training parameters. The results are presented in Table 3.10.

We observe that, similarly to the case when $R = 64$, larger compute budgets imply larger optimal values of G. Note that the values for the 10^{th} and 90^{th} percentiles form larger intervals in this case, as in this part, we run fewer experiments and keep shorter

N	D	G
16 x 100M	(10.29B, 17.73B)	(8 , 16)
16 x 1B	(53.74B, 103.54B)	(16, 32)
16 x 3B	(106.22B, 261.04B)	(16, 32)
16 x 7B	(177.65B, 511.43B)	(16, 32)
16 x 70B	(721.60B, 3.22T)	(32, 64)
16 x 300B	(1.73T, 10.69T)	(32, 64)
16 x 1T	(3.60T, 28.22T)	(32, 128)

Table 3.10: 10^{th} and 90^{th} percentiles estimated via bootstrapping data for $R = 16$.

training durations. However, we believe that this preliminary study provides a valuable addition to the results in the main part.

3.8 Discussion

Extreme Granularity. In Section 3.4, we argue that model performance improves with increasing granularity. This postulate largely aligns with the empirical findings of our study. Nonetheless, at exceedingly high granularity levels, such as $G = 64$, in models characterized by $d_{\text{model}} = 256$ and $R = 64$, there is an observable decline in performance. This phenomenon is particularly evident in scenarios where the number of parameters in the routing mechanism exceeds the active parameters in actual experts. Additionally, as described in Section 3.5, the utility of such high granularity is predominantly restricted to models of substantial size. In alignment with the principles outlined in Hoffmann et al. (2022), this research focuses more on findings that can be broadly applied rather than examining these corner-case situations in detail. However, it is hypothesized that the efficiency of models with significantly high granularity could potentially be enhanced through careful expert initialization or modifications to the routing algorithm. These ideas are set aside to be investigated in future studies.

Including R in the formula. Another potential advancement would be to unify all the factors N, D, G, and R into a single formula. While this approach would enable a more detailed analysis of the relationships between coefficients, it would be challenging to recommend the optimal configuration using only FLOPs. Larger values of R generally lead to better performance but also require additional memory. Consequently, the choice of expansion rate may be heavily dependent on the available hardware configuration. A detailed study of these factors is left for future work.

Choosing Granularity. Table 3.4 (for expansion rate 64) and Table 3.9 (for expansion rate 16) list the optimal granularity values for various compute budgets. Figure 3.1(a) presents an alternative representation for $R = 64$. We observe that the optimal granularity values are generally similar between different expansion rates. We can generally provide the following guidelines:

- The standard value of $G = 1$ is almost never optimal.
- For a reasonable default value of G, refer to Table 3.4 and Table 3.9. For constant N or D, one can calculate the optimal G and the trade-off between predicted loss and

training FLOPs directly from our scaling laws using the coefficients from Table 3.2.

- The exact optimal value of granularity may differ slightly from our setup, but based on other works on scaling laws, we expect only slight differences.

3.9 Conclusions

This chapter introduced a novel hyperparameter, granularity (G), and underscored the significance of adjusting it for optimizing the efficiency of experts within MoE models. A central finding of this research is that a standard granularity of $G = 1$ is suboptimal across a broad range of FLOPs. Therefore, we recommend using higher granularity values to enhance MoE model performance and efficiency. Simultaneously, this chapter emphasizes the importance of varying training duration for compute-optimal settings. Consequently, both granularity and variable training length are incorporated into new scaling laws. These laws confidently demonstrate that MoE models consistently outperform dense Transformers in terms of efficiency and scaling. We shed new light on the scaling laws applicable to MoE models and provide practical guidance for improving computational efficiency in Large Language Models. These insights are critical for the development and optimization of large-scale language models, marking a significant advancement in the field.

Since the efficiency gains of integrating Mixture of Experts into Large Language Models will only grow with scale, further improvements to those methods are of utmost importance. While training larger models is usually less stable, training MoE models causes yet additional instabilities, as often evidenced by the need to decrease the learning rate when using MoE (Rajbhandari et al., 2022). The significant cause of this instability lies in the fact that the MoE layer is not continuous. This discontinuity not only prevents gradients from propagating when a particular expert is not processing a particular token, but it may also cause gradients to explode when the same expert unexpectedly processes the token due to randomness in the training batch. While increasing the granularity decreases the discontinuity, a more principled approach is in order. Such an approach will be presented in Chapter 4.

Chapter 4

Mixture of Tokens: Continuous MoE through Cross-Example Aggregation

Non exiguum temporis habemus, sed multum perdidimus.
"It is not that we have a short space of time, but that we waste much of it."
— Seneca, *De Brevitate Vitae*

4.1 Introduction

Sparse activations in MoE models are made possible with a *router*, a small network that selects the best experts for each token. This makes the output of an MoE layer discontinuous with respect to its parameters, as choosing a subset of the experts for each token is done with a discrete *top-k* operation. The discontinuity and the resulting fluctuations of the router's decisions have been shown to hurt training efficiency (Dai et al., 2022; Chi et al., 2022) and are hypothesized to be a source of training instability in large MoE models (Mustafa et al., 2022; Puigcerver et al., 2023). On the other hand, existing continuous MoE architectures involve trade-offs, including the inability to scale (Muqeeth et al., 2023; Hazimeh et al., 2021) or incompatibility with autoregressive decoding (Puigcerver et al., 2023).

This chapter introduces Mixture of Tokens, a novel continuous Transformer architecture closely related to sparse Mixture of Experts. Similar to MoE, it can support large parameter counts without incurring significant costs in FLOPs. The core idea behind our design is for each expert to process not individual tokens separately but their combined representation.

This technique results in a continuous model that avoids the top-k operation. It requires no additional techniques commonly needed in existing MoE designs (both sparse and continuous), such as load-balancing losses, calculating solutions to optimization problems, or non-homogeneous training schedules (Hazimeh et al., 2021; Dai et al., 2022). It is capable of scaling the parameter counts akin to sparse MoEs and is compatible with autoregressive language modeling and generation. Our best MoT models not only achieve a 3$\times$ increase in training speed over dense Transformer models in language pretraining but also match the performance of state-of-the-art MoE architectures. Additionally, a close connection between MoT and sparse MoEs is demonstrated through a novel technique we call *transition tuning*.

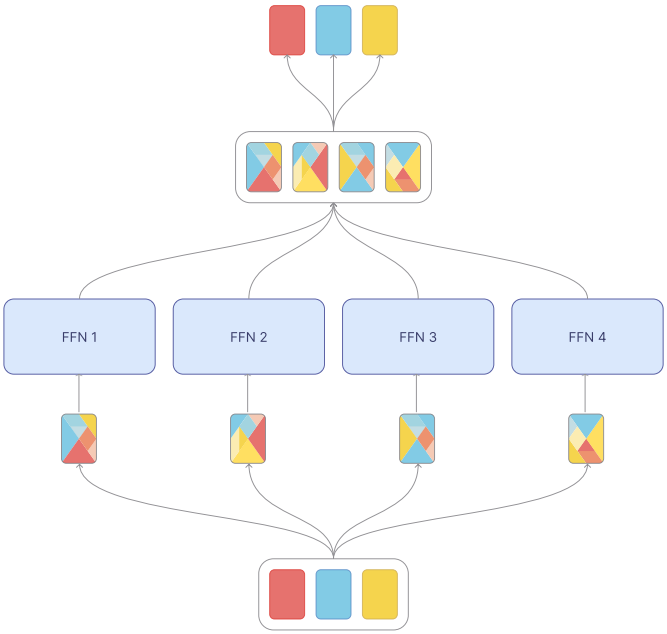


Figure 4.1: Mixture of Tokens: Each expert receives a unique mixture of tokens in the group. Mixing weights are determined by the router, which is a fully connected layer (omitted for clarity). For a given token, its update is a linear combination of expert outputs, with the coefficients equal to the token's original mixing weights for each expert. Compare with Figure 3.2 in Chapter 1, which presents a standard feed-forward and standard token-choice MoE.

4.2 Related Work

In this section, we provide an overview of the work related to MoT specifically. For a full discussion on MoE, refer to Chapter 1. We will introduce the Mixture of Tokens architecture in Section 4.3 and provide a detailed comparison between MoT and related methods in Section 4.3.3.

4.2.1 Continuous Mixture of Experts

Continuous architectures serve an important role within the field due to their flexible and efficient nature. Hazimeh et al. (2021) were pioneers in introducing them in MoE, presenting continuous techniques for calculating the encodings of the choice of an expert. In another approach, Muqeeth et al. (2023) proposed a method where they merge experts based on the weights of the router network. In a recent advancement, Puigcerver et al. (2023) proposed a continuous variant of MoE for the Vision Transformer, where patches are mixed only within each image.

4.2.2 From Hard to Soft Methods

From the very beginning of the Deep Learning field, there has been a movement from discrete functions toward continuous ones. The first perceptron (McCulloch & Pitts, 1943) used "all-or-none" activation, supposedly to align with propositional logic. This was later improved with soft activation functions, enabling gradient descent and multi-layer neural

networks. Similarly, soft attention introduced in Bahdanau et al. (2016) enabled RNNs to look at arbitrary input from the past while retaining the ability to learn selection using standard gradient descent. This contrasts with hard attention, which requires, for example, reinforcement learning techniques. While hard attention could perform on par with soft attention (Xu et al., 2015; Zohourianshahzadi & Kalita, 2021), soft attention, with its simplicity of training, presented better trade-offs and was later used as the basic building block of the Transformer (Vaswani et al., 2017).

Mixture of Experts, introduced into Deep Learning by Shazeer et al. (2017), seems like a naturally discrete function—after all, the expert either processes a given token or it doesn't. However, just like moving from hard to soft attention, an expert in MoE can "attend" to a mix of tokens taken as a weighted average, resulting in a smooth, continuous model and enabling more stable training.

4.3 Mixture of Tokens

The goal of this chapter is to devise an efficient, continuous architecture that retains the scalability of Mixture of Experts while omitting the top-k operation that limits a token's exposure to different experts. An intuitive way to achieve this is to route all tokens to all experts; however, this approach is computationally infeasible for large-scale pretraining. To combat this constraint, the method explored in this chapter considers what happens not to an individual token but to a whole group of tokens instead. The main contribution of this chapter is the observation that allowing an expert to dynamically produce a continuous representation of the entire group of tokens, which is more lightweight to process than each token individually, yields positive results.

More specifically, in our design, an input batch is divided into groups of tokens, and each group is processed independently. Given a group and a single expert, a scalar weight is produced for each token. The weights are then normalized and used to compute a linear combination of the tokens, which serves as the expert's input. The experts' outputs are used for token updates as follows: for each input token, its update is a linear combination of expert outputs, with the token's mixing weights for each expert as coefficients. This has an efficient vectorized implementation, where all meaningful computations are performed with batched matrix multiplications. A diagram of our method is presented in Figure 4.1.

Algorithm 1 Mixture of Tokens Layer

for each E in experts **do**:
  $\text{weights}_E = \text{Softmax}(\text{Linear}(\text{tokens}))$
  $\text{mix} = \sum_i \text{token}_i * \text{weights}_{i,E}$
  $\text{output}_E = E(\text{mix})$
end for
for each i **do**
  for each E **do**
    $\text{update}_i = \sum_E \text{output}_E * \text{weights}_{i,E}$
  end for
end for

To understand why this method is scalable, it is useful to examine the relationship between the number of tokens in a group and the number of experts. Essentially, if the two are equal, the total computation done by experts is the same as in the case of top-1

routing. Consequently, MoT enjoys the same parameter scaling benefits as observed in MoE, which we confirm empirically in Section 4.4.3.

4.3.1 Granularity in MoT

Building on the design described above, we experiment with feeding more than one mixture into each expert. If done without any further modifications, this would mean a linear increase in computation costs for each extra mixture processed. In order to avoid this extra cost, MoT uses more experts, but each expert has a proportionally reduced hidden dimension. This is the granularity of experts introduced in Chapter 3. In this way, each mixture is processed by a small expert, and the layer's total number of parameters, as well as the number of FLOPs used by all experts, remains the same. We find this design brings consistent improvements as the number of processed mixtures increases, consistent with results on sparse Mixture of Experts.

4.3.2 Token Groups in MoT

The question of how token *groups* are decided within a batch is crucial for compatibility with autoregressive training and inference. The main insight here is that tokens from the same sequence cannot be placed in one group, as the mixing operation would result in an information leak. Due to this restriction, MoT groups tokens from different examples based on position in the sequence. In effect, all tokens within a group have the same position in their respective sequences. As mentioned before, in order to keep the number of FLOPs per token constant, an increase in the number of experts means an equal increase in group size. An illustration of how grouping is done within a batch of tokens is shown in Figure 4.2.

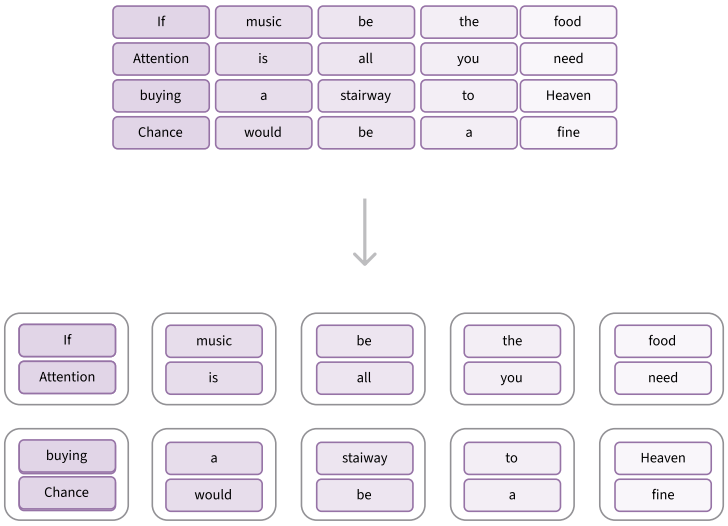


Figure 4.2: Each group consists of tokens with the same position in a sequence. In this example, the group size is 2. Note that the maximum possible group size is equal to the batch size.

4.3.3 Comparison with Other Mixture of Experts Architectures

Scaling. The technique featured in Hazimeh et al. (2021) is based on a continuously differentiable sparse top-k router, which is a major advantage compared to the common top-k gating. However, this approach requires that all experts are utilized during a portion of the training, rendering it computationally prohibitive for models with large numbers of experts. The architecture based on merging experts proposed in Muqeeth et al. (2023) also presents an attractive continuous alternative to top-k gating, yet the cost of merging all experts again scales linearly with the number of experts. To combat this, the technique is applied once per sequence, which limits the expressive power of the final model.

Training Stability. Lepikhin et al. (2020) report instabilities during the training of large MoE models, stemming from inaccuracies when calculating router weights in low precision. To stabilize the training, they resorted to using full precision. Fedus et al. (2022) made progress by using mixed precision when training MoE, implementing selective high precision for gating. When comparing Lepikhin et al. (2020) and Fedus et al. (2022) to MoT, an advantage of our technique emerges: it is more robust to training in lower precision than other methods. We conjecture that this robustness is due to the merging mechanism being less susceptible to rounding errors than gating in sparse MoEs.

Token Dropping. Token dropping is a phenomenon where tokens do not receive an update from any expert. This can occur because an expert was chosen by too many tokens in a batch (Fedus et al., 2022; Lepikhin et al., 2020; Zoph et al., 2022), or, in the case of routing experts to tokens, when a token was not chosen by any expert (Zhou et al., 2022). Existing techniques that combat this phenomenon provide a partial solution, but the problem remains. In contrast, tokens in MoT are part of every mixture produced within their group; hence, they always receive an update.

Auto-Regressive Decoding. Mixture of Tokens is based on the notion of merging tokens prior to being processed by an expert. An encoder-only design of a similar flavor is featured in concurrent work (Puigcerver et al., 2023). This technique is based on merging patches within an image for vision models. The crucial difference between this technique and MoT is that MoT is compatible with autoregressive training and inference.

4.4 Experiments

The focus of this chapter is to investigate the efficiency of Tokens on autoregressive language modeling. To measure model quality, we pretrain models for a fixed number of tokens and compare final perplexity in accordance with existing MoE literature (Du et al., 2022; Fedus et al., 2022). In all experiments, the models are trained on the C4 dataset (Raffel et al., 2019) and use the GPT-2 tokenizer—as in Chapter 3. Unless specified otherwise, we use mixed precision, where all heavy computation is done in bfloat16, whereas the optimizer state and the master weights are kept in full precision. To study the stability of our model, we experiment with training fully in reduced precision.

Our main result is a substantial speed-up of MoT models compared to dense Transformers (Figure 4.6) and results comparable to sparse MoEs (Figure 4.5). What follows is an analysis of scaling properties of the MoT architecture with respect to the number of parameters (Figure 4.3) and the number of mixtures sent to each expert (Figure 4.4). We investigate the model's performance in low precision in order to simulate training instability and find that MoT is less susceptible to instabilities stemming from low-precision training. Lastly, we show the connection between MoT and MoE by spending

an additional fraction of pretraining compute to effectively turn a MoT model into a Token Choice model (Section 4.4.5).

4.4.1 Model Architecture

The basis of our experiments is a decoder-only Transformer modeled after GPT-2 (Radford et al., 2019). We conduct experiments on two model scales: a 77M Medium model and a 162M Base model (refer to Appendix 4.4.2 for hyperparameters and training details). To create a Mixture of Tokens (MoT) model, we replace the second half of feed-forward layers in the Transformer with MoT layers. Similar to MoE models, the FLOPs and parameter counts in MoT are decoupled. We denote the model architecture by its dense counterpart in terms of the number of FLOPs and, separately, the number of experts (or, equivalently, group size).

Thus, a MoT-Medium/32E model utilizes the same number of FLOPs as a Medium (77M) Transformer model but employs 32 experts in MoT layers.

As described in Section 4.3.1, Medium/32E/4 represents a model that uses MoT layers with $32 \cdot 4$ small experts, totaling the same number of parameters as 32 normal experts—an expansion rate R of 32 and granularity G of 4.

In addition to using the Transformer as a baseline, we compare MoT against Token Choice (Fedus et al., 2022) and Expert Choice (Zhou et al., 2022) as sparse MoE baselines. As Expert Choice is sensitive to the batch size, to avoid discrepancies between training and inference, we group tokens prior to routing in training Expert Choice models.

4.4.2 Training Hyperparameters

All models were trained using mixed precision unless explicitly stated otherwise. We conducted all experiments using a batch size of 256 and a context length of 256 for 150K training steps, resulting in a total of 10B training tokens. The AdamW optimizer was used with default hyperparameters. When necessary, a Fully Sharded Data Parallel approach from PyTorch was utilized to parallelize training across multiple machines. Learning rates were independently tuned based on model size and architecture. The optimal learning rate for Transformers was 1e-3 for Medium and 4e-4 for Base models, while for both MoT and MoE, they were 7e-4 for Medium and 2e-4 for Base.

Model	Experts	Expert size	Group size	Total params	Blocks	d_{model}	d_{ff}	#att. heads
Transformer-Medium	-	-	-	77M	8	512	2048	8
MoT-Medium/32E	32	2048	32	336M	8	512	-	8
MoT-Medium/32E/8	256	256	32	337M	8	512	-	8
Transformer-Base	-	-	-	162M	12	768	3072	12
MoT-Base/32E	32	3072	32	520M	12	768	-	12
MoT-Base/64E/16	1024	192	64	977M	12	768	-	12

Table 4.1: Training hyperparameters. The table provides example models featured in experiments. All remaining models can be derived from this table.

4.4.3 Scaling Results

Mixture of Tokens models demonstrate strong scaling properties with respect to the number of parameters. As seen in Figure 4.3, increasing the number of experts in MoT layers while using the same compute budget yields consistent improvements. All MoT models are a strict improvement over the Transformer. The figure also features an ablation experiment, where the mixing weights are fixed to $1/n$, where n is the group size. This corresponds to a uniform mixing strategy. The performance of that model clearly suffers, confirming that MoT layers learn non-trivial mixing strategies.

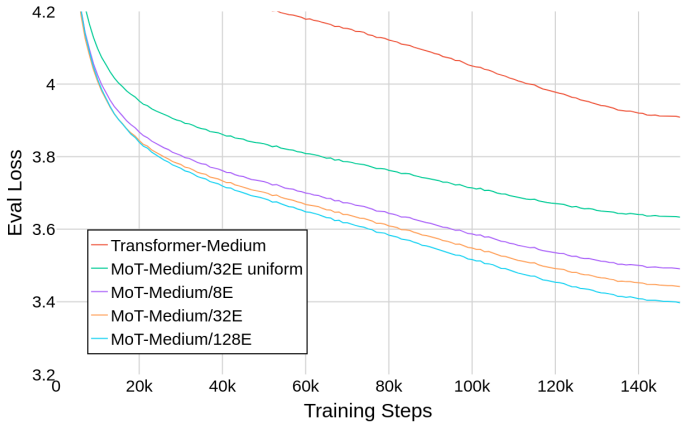


Figure 4.3: Scaling with respect to the number of parameters. Also featured are the Transformer baseline and an MoT model with a non-learnable, uniform routing strategy.

The increased number of token mixtures (granularity) described in Section 4.3.1 is another axis of scaling for MoT models, again exhibiting consistent improvements. We hypothesize that this phenomenon is due to two mechanisms. First, the model is simply more expressive with a larger number of smaller experts. Second, the model can allocate its focus (the mixing weights) more flexibly to more important tokens while reducing updates for trivial ones.

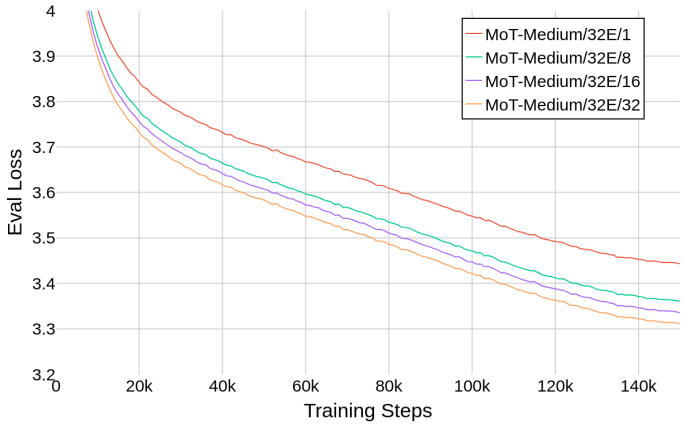


Figure 4.4: Results of increasing granularity in MoT show that findings from Chapter 3 generalize to different MoE designs.

4.4.4 Comparison with the Transformer and Sparse MoEs

Crucially, the performance of Mixture of Tokens is comparable to that of the strong Mixture of Experts baselines (Figure 4.5). An increased number of mixtures allows it to compete with both Expert Choice and Token Choice architectures. As sparse routing is hypothesized to contribute to training instabilities in large sparse models, Mixture of Tokens, being continuous, presents a promising alternative. To investigate training instabilities at the scale we experiment on, we trained models fully in bfloat16, as opposed to the mixed precision used in all other experiments. The results confirm that MoT is more resistant to lower precision training: as the precision of training decreases, the performance of Expert Choice drops below that of Mixture of Tokens, despite the former attaining better perplexity using mixed precision. We find this to be evidence of the architecture's potential for stable training at higher model scales. See Table 4.2 for details.

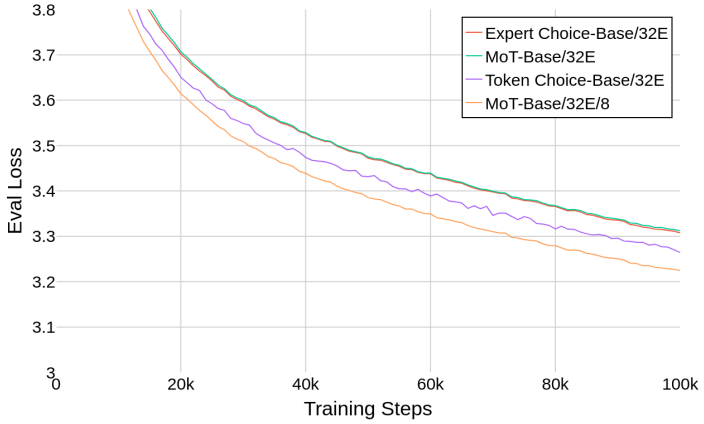


Figure 4.5: Comparison of MoT and sMoE architectures. An increased number of smaller experts allows MoT to match the performance of the best sMoE model. Due to computational constraints, the models were trained for 100K steps.

	MoT-Medium/32E	Expert Choice-Medium/32E
Mixed Precision	3.442 (± 0.002)	**3.420 (± 0.002)**
bf16 only	**3.661 (± 0.007)**	3.728 (± 0.044)

Table 4.2: Lower precision training result loss comparison. MoT is better in the bfloat16-only setting. Learning rates were separately tuned in lower precision for both EC and MoT. Results are averaged over 3 random seeds.

Lastly, we combine our findings on MoT scaling properties to train our most efficient MoT model and compare it to the Transformer baseline (Figure 4.6). The result is a model that attains the final loss of the baseline in a third of the training steps. This represents a 3× improvement in terms of the compute budget.

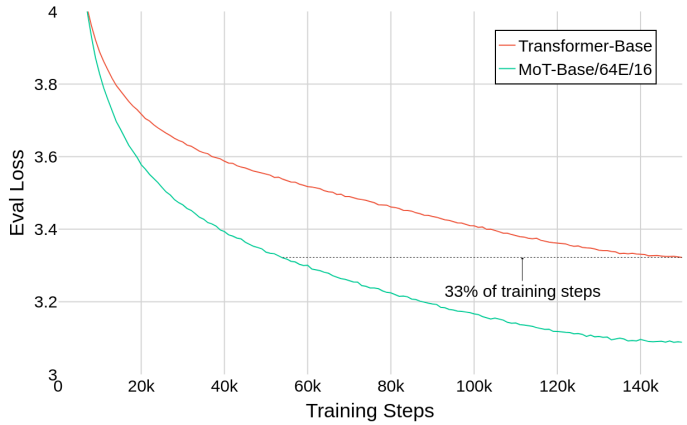


Figure 4.6: Our best MoT model reaches the final loss of the baseline in just 33% of the compute budget.

4.4.5 Transition Tuning

Mixture of Tokens suffers from a drawback common to many MoE designs, namely, it does not support unbatched inference. This is a direct consequence of its design—in the forward pass, it groups several tokens from different examples in the batch. With the growing adoption of Large Language Models on consumer hardware (Touvron et al., 2023; Cerisara, 2023), this lack of support could hinder the architecture's wider adoption. While Mixture of Tokens with a group size of one is technically possible, to keep FLOPs constant, the layer would have to reduce trivially to a standard Transformer MLP.

To address this, we demonstrate that the weights learned by Mixture of Tokens can be used to directly initialize a Token Choice model of the same specifications (number of experts and expert size). The layer responsible for producing mixing weights is used to initialize the sparse router. To mitigate the difference in performance caused by this architectural change, we train the entire new model (without freezing any weights) for 10% of the total pretraining steps of the original model to recover the original model's performance (measured in eval loss). We call this technique *transition tuning*. This way, it is possible to train with Mixture of Tokens and enjoy unbatched generation at inference time. We hypothesize that this pipeline would be especially attractive in setups where having parts of the model train in higher precision is impossible, e.g., on specialized, low-precision hardware. The results are presented in Figure 4.7.

4.5 Limitations and Future Work

With the strong performance of MoT on medium-sized models, an obvious next step is to train larger models. This would present an opportunity to validate the stability results on larger models, where training instabilities are more common.

As with most Mixture of Experts models, the memory footprint of MoT layers is substantial. Scaled models require large amounts of RAM on specialized hardware for training, which makes their adoption expensive. To this end, an attractive future direction would be to investigate model distillation with Mixture of Tokens models.

Lastly, both training and inference with MoT mix different examples within a single batch. This mixing of tokens from different sequences and the requirement of performing

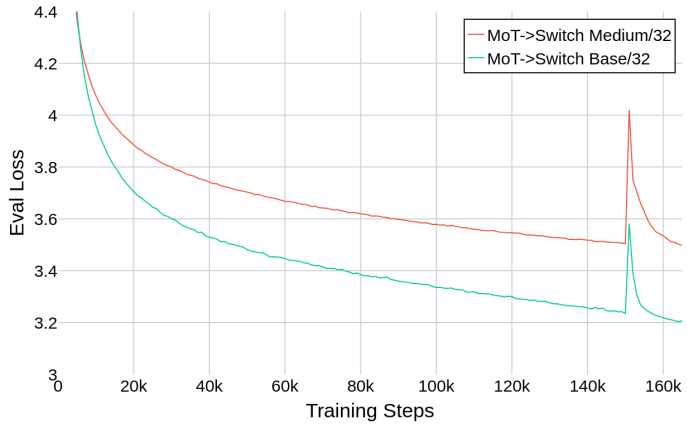


Figure 4.7: Transition tuning: The first 150K steps of the model are done using a Mixture of Tokens architecture. Then, a new Token Choice model is initialized with weights taken from the MoT model, and the model trains for a further 15K steps to recover performance. The spike in loss is due to the sudden change of architecture.

batched inference may be undesirable in some use cases. While performing unbatched inference is always inefficient with LLMs, as the memory throughput to access model weights becomes the bottleneck, unbatched inference still finds its uses. Even though transition tuning solves this problem, exploring different inference strategies might bring new insights.

4.6 Conclusions

In this chapter, we presented the Mixture of Tokens, a novel continuous Mixture of Experts architecture compatible with autoregressive decoding. This architecture scales to model sizes similar to sparse Mixture of Experts and matches its performance, and is more resistant to training instabilities due to lower precision training. Moreover, we introduced transition tuning, a technique of initializing an MoE model with another pretrained MoE model of different architecture, and showed that the new model attains the performance of the original one using a fraction of the compute budget.

In previous chapters, we explored efficient training and inference of LLMs when scaling up their parameter count and dataset sizes. Additionally, we increased the stability of training LLMs. However, there is one more important trend in LLMs nowadays: scaling up the context size, that is, the length of a single sequence. Given the growing capabilities of these models, it is reasonable to assume that they can utilize more information/context in each example. An example is LSH attention used in the Terraformer in Chapter 2. While many works have addressed the efficient processing of long context, models often poorly utilize this abundance of information. In the next chapter, we explore ways to increase the efficiency of such context utilization.

Chapter 5

Structured Packing in LLM Training Improves Long Context Utilization

Je n'ai fait celle-ci plus longue que parce que je n'ai pas eu le loisir de la faire plus courte.
"I have made this letter longer than usual because I lacked the time to make it shorter."
— Blaise Pascal, *Lettres Provinciales*

5.1 Introduction

Together with expanding sizes and training durations of Large Language Models, the context lengths they can technically process also grow. This is important, especially for processing extremely long documents, such as scientific publications, books, or entire code repositories. While nearly every modern LLM is also a Long-Context Language Model (LCLM), their ability to process long context is not always as effective as one hopes. Indeed, several studies have highlighted an important limitation: when processing prompts composed of multiple documents, Long-Context Language Models (LCLMs) frequently encounter difficulties in accurately extracting relevant information (Tworkowski et al., 2023; Liu et al., 2023a; Shi et al., 2023a). Additionally, they typically find it challenging to utilize information from the middle of their inputs (Liu et al., 2023a), even on simple synthetic retrieval tasks (Li et al., 2023a). Understanding these issues is vital for advancements in LCLM technologies and calls for systematic research.

In this work, we take a step towards better context utilization in LCLMs. We focus on training data, keeping other components, such as the architecture and training objectives, unchanged. The broad question is *how to organize training data to enhance long context capabilities*? Such perspective has received some attention recently (Levine et al., 2022; Chan et al., 2022; Shi et al., 2023b), but the problem is far from being solved. The central finding of this work is that *structuring training data to increase semantic interdependence is an effective strategy towards better long context utilization.* We achieve this by introducing and evaluating Structured Packing for Long Context (SPLiCe), a method for creating training examples by using retrieval (e.g., BM25, Contriver or repository structure) to collate mutually relevant documents into a single training context.

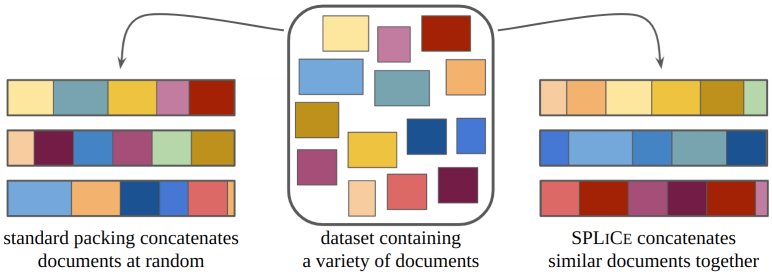


Figure 5.1: Training examples generated by the standard sampling procedure vs proposed by SPLiCe. Similar colors indicate related documents, which could be found using a retrieval method (e.g., BM25 or contriver) or metadata (e.g., repository structure for code).

We empirically validate SPLiCe across models of varying sizes, showing that fine-tuning of OpenLLaMA 3Bv2, 7Bv2 (Geng & Liu, 2023) and CodeLlama 13B (Rozière et al., 2023) with mere 2B–6B tokens already brings substantial improvements in handling long context information in downstream tasks that require retrieval and in-context learning. These tasks include Qasper (Dasigi et al., 2021) from SCROLLS (Shaham et al., 2022), lost-in-the-middle benchmark (Liu et al., 2023a), HotPotQA (Yang et al., 2018), TREC (Li & Roth, 2002; Hovy et al., 2001), and DBpedia (Lehmann et al., 2015). We perform a comprehensive study of the design choices and properties of SPLiCe, showing, in particular, that the acquired long context capabilities transfer between modalities. For instance, training on programming code enhances performance on natural language tasks. SPLiCe also helps to retain and in some cases even improve performance on short context benchmarks like GSM8K (Cobbe et al., 2021), MMLU (Hendrycks et al., 2021) and TREC (Li & Roth, 2002; Hovy et al., 2001).

5.2 Related Work

There is an increasing number of works aiming to study the role of data in LLM training in detail. For instance, (Levine et al., 2022) developed a theory and demonstrated empirically that incorporating non-adjacent but semantically related sentences in training examples leads to better sentence embeddings and improves open-domain question-answering performance. Another work (Gu et al., 2023) introduced a pretraining framework grounded on the idea that text documents often include intrinsic tasks. They showed that this approach substantially boosts in-context learning. Additionally, there is existing work on training long-context language models using repository-level code data, such as (Wu et al., 2022). Work (Chan et al., 2022) identifies the training data's distributional properties that affect transformer models' in-context capabilities. Similarly, Han et al. (2023) constructs small-scale data using an iterative gradient approach and shows that such data improve in-context performance.

Our methodology diverges from these works in several key ways. Firstly, we focus on the document-level context during the training phase and long context performance, as opposed to sentence-level (Levine et al., 2022) or paragraph-level (Gu et al., 2023) granularity. We demonstrate the efficacy of our approach in large-scale language modeling, specifically with OpenLLaMA 3B, 7B, and CodeLlama 13B. Secondly, we construct a tree structure of related documents through BM25/Contriever-MSMARCO retrieval and linearize this structure to form long-context examples. This allows for greater control over

example coherence compared to relying solely on natural data structures like repository-level code. The gradient-based method presented in Han et al. (2023) can be broadly associated with retrieval used in this chapter; however, it differs in scale and granularity.

Concurrently to our research, (Shi et al., 2023b) showed a method for preparing the training data that closely resembles SPLiCe with default settings ($k = 1$ and the identity as `Order`). The key differences are that (Shi et al., 2023b) focuses on training from scratch while we aim to achieve long context capabilities with short and cheap fine-tuning. Next, we use longer context lengths, up to 32K tokens, whereas (Shi et al., 2023b) utilizes 8K. Finally, we provide detailed analysis regarding the design choices, such as the benefits of data reordering in Section 5.4.3 and different values of k in Section 5.4.3.

5.3 Method

SPLiCe is a method for constructing training examples that enhance the LLMs' long context utilization. This translates to improved performance on a range of tasks like in-context learning, question answering, in-context information retrieval, and long-context language modeling.

Algorithm 2 SPLiCe training example construction

Input:
D: document corpus
k: breadth hyper-parameter
L: maximum length of returned training example
`RETRIEVE`: retrieval method to use, e.g., BM25
`ORDER`: ordering method, e.g., identity, or random shuffle

Output: training example consisting of concatenated documents
$d_r \sim D$ {Sample the root document}
$D = D \setminus \{d_r\}$
$C = [d_r]$
$Q =$ `empty queue`
Q.`PUSH`(d_r)
while $Q \neq \emptyset$ and `len`$(C) \leq L$ **do**
  $d = Q$.`POP`$()$
  $d_1, \ldots, d_k =$ `RETRIEVE`(d, k)
  {Retrieve top-k most similar documents to d using a selected method, e.g., BM25}
  for each d_i in $d_1, \ldots, d_k$ **do**
    if $d_i \in D$ **then**
      {`RETRIEVE` uses a precomputed index and may return documents that are already in C}
      $C = C$.`APPEND`(d_i) {Append d_i to C}
      Q.`PUSH`(d_i)
      $D = D \setminus \{d_i\}$
    end if
  end for
end while
return `TRIM`(`ORDER`$(C), L$)

Rationale and Intuitions. Capturing long-range dependencies is believed to enhance language modeling and retrieval-augmentation (Borgeaud et al., 2022). However, it is an open question how to achieve such benefits in pre-training or fine-tuning. The primary difficulty comes from long-range dependencies being rare in training data (de Vries, 2023) and diminishing with distance. Thus, it is unlikely that a model will learn to utilize long context without more explicit guidance.

Recent studies indicate that structuring data, i.e., going beyond the i.i.d. paradigm, might be beneficial or even necessary to achieve good long-context performance. (Levine et al., 2022) develops a theory showing that the trained model establishes stronger dependencies between text segments in the same training example. Chan et al. (2022) indicates that specific distributional properties of the training data have a significant impact on the model's in-context performance. Finally, concurrently to our work (Shi et al., 2023b) shows that pre-training on structured data improves the performance of LLMs.

SPLiCe follows these intuitions. Namely, it constructs training examples by collating mutually relevant documents to increase the dependency density, thus allowing the model to learn to utilize long context. We provide additional insights into the distributional properties of SPLiCe created data at the end of this section.

Structured Packing for Long Context (SPLiCe) SPLiCe starts by picking a random document from the dataset to create a root of a tree and continues in a breadth-first manner, each time appending top-k similar documents from the corpus. The final sequence consists of the tree flattened according to a certain tree traversal strategy; see Algorithm 2.

The hyperparameter k introduces flexibility, enabling interpolation between different modes of retrieval. Specifically, when $k = 1$, SPLiCe simulates a long document by creating a path of related examples. For larger k values, SPLiCe generates examples akin to those used in retrieval-augmented models, e.g., (Borgeaud et al., 2022). It is essential for the dataset to be well-curated to ensure the efficient functioning of SPLiCe. For instance, maintaining a low duplication rate encourages the model to leverage long context beyond merely copying. In this work, we employ the StarCoder (Li et al., 2023b) dataset, which has been deduplicated using the pipeline described in (Allal et al., 2023).

Many possible retrieval methods can be used with SPLiCe (RETRIEVE function in Algorithm 2). In our experiments, we test the following:

- SPLiCe REPO: based on additional meta-information about the data, that is the repository structure of the code (REPO): we concatenate files using a depth-first search algorithm on the directory structure, that is files from the same directory are grouped together. A similar method has been pioneered by (Wu et al., 2022) and proposed in (Shi et al., 2023b) as an interesting future direction.
- SPLiCe BM25: based on BM25 (Robertson & Zaragoza, 2009; Bassani, 2023), a standard retrieval method that uses a bag-of-words approach to rank documents based on their similarity to a query.
- SPLiCe CONT: based on Contriever-MSMARCO (CONT) (Izacard et al., 2022), a recently proposed retrieval method that uses a transformer to rank documents based on their similarity to a query.

Baselines One standard approach, commonly used in LLM training pipelines, is to randomly sample documents from the dataset and concatenate them to make training

examples of a desired context. This is known as example packing (Brown et al., 2020). Another popular approach restricts sampled documents to the same meta-class (for example, Wikipedia article, source code) (Groeneveld et al., 2024; Zhao et al., 2024). We compare SPLiCe against both.

SPLiCe computational efficiency Given the dataset sizes used in training LLMs, computational efficiency plays a non-trivial role. SPLiCe Repo is the fastest and easiest to implement but requires additional directory structure, i.e., it does not apply to general web data. SPLiCe BM25 uses a bag of words BM25 method that lacks deeper semantic encoding. SPLiCe Cont requires calculating embeddings for each document and retrieval based on the vector inner-product. The latter step can be done efficiently using fast approximate inner-product search, e.g., Faiss (Johnson et al., 2017).

5.4 Experiments

In this section, we show that SPLiCe improves the long context performance of large-scale language models. To this end, we use 3B, 7B, and 13B parameter models and evaluate on tasks that test in-context learning (Section 5.4.3), question answering (Section 5.4.3), and in-context information retrieval capabilities (Section 5.4.3). Additionally, we provide perplexity results in Section 5.4.3. Next, we show that SPLiCe can improve the core model capabilities by testing the short context performance, see Section 5.4.3. Finally, in Section 5.5, we tune over 30 medium (270M parameters) models using different data mixtures and SPLiCe parameters, to analyze its design choices.

An important finding of our work is that presented improvements occur during a relatively short, compared to pre-training, fine-tuning. To be more precise, 3B models were tuned on 5.4B tokens, whereas 7B and 13B models were tuned on 2B tokens. The 3B and 7B start from OpenLLaMAv2 (Geng & Liu, 2023) that was pre-trained on 1T tokens, the 13B parameter models start from CodeLlama (Rozière et al., 2023) that was pre-trained on 2.5T tokens.

5.4.1 Experimental Setup

For 3B experiments, we continue training on a 50/50 mixture of RedPajama, prepared in the standard way, and C prepared using SPLiCe BM25. For 7B and 13B ones, we continue the training on a 50/25/25 mixture of RedPajama (50) prepared in the standard way, StackExchange (25) and C (25) prepared using SPLiCe BM25. StackExchange is part of RedPajama (TogetherComputer, 2023), and C data come from StarCoder (Li et al., 2023b). Including the standard RedPajama aims to prevent the model from overfitting to artificially created long documents. We refer to Section 5.4.2 for a detailed description of the model architectures.

We train with 32K context length. We use a batch size of 256K tokens per step and the learning rate of 1.5e−5 with linear warmup and cosine decay, following (Geng & Liu, 2023). For 13B model we use batch of 512K tokens per step.

Regarding the context extensions method, we use FoT (Tworkowski et al., 2023), and the CodeLlama (CL) (Rozière et al., 2023) approach. More explicitly, we test **eight models**:

$$\{\text{3B FoT, 7B FoT, 7B CL, 13B FoT }\} \times \{\text{SPLiCe, Baseline}\},$$

where Baseline denotes the standard (example packing) data preparation method where context is created by sampling random documents from the corpus and separating them

with `BOS`/`EOS` tokens. If not stated otherwise, in SPLiCe we use $k = 1$ and the identity permutation as `Order` in the Algorithm 2.

5.4.2 Architecture and Hyperparameters

The architecture of our models is based on LLaMA (Touvron et al., 2023), and the architectural details can be found in Table 5.1. Briefly speaking, our architecture is a standard decoder-only Transformer, with a few standard changes. Firstly, we perform RMSNorm before the input of both the attention and feed-forward modules. Secondly, we use the LLaMA Feed-Forward module. Additionally, we use Rotary Position Embedding (Su et al., 2021). For context extension, we use Focused Transformer (FoT)(Tworkowski et al., 2023), CodeLlama (Rozière et al., 2023) (CL) and YaRN (Peng et al., 2023b). Table 5.2 presents the details about both standard and long-context pretraining. We use AdamW as an optimizer, with $\beta_1 = 0.9$ and $\beta_2 = 0.95$.

Parameter/Model Size	**270M**	**3B**	**7B**	**13B**
Vocab Size	32000	32000	32000	32016
Embedding Size	1024	3200	4096	5120
Num Attention Layers	12	26	32	40
Num Attention Heads	8	32	32	40
Head Size	128	100	128	128
MLP Hidden Size	4096	8640	11008	13824
FoT Context Extension Layers	[6, 9]	[6, 12, 18]	[8, 16, 24]	[12, 24, 36]

Table 5.1: Architecture details. Focused Transformer context extension is applied in continued pretraining for 32K context and evaluation.

Each tuning experiment of 270M model was done using either a TPUv3-8 or TPUv4-8 machine and took around 10 hours. Tuning a 3B parameter model for 5.4B tokens with FoT took 40 hours on TPUv3-128. The 7B and 13B models were tuned on TPUv3-128.

Stage	**Parameter/Model Size**	**270M**	**3B**	**7B**	**13B**
Pretraining	Context	2K	2K	2K	16K
	Tokens	6.3B	1T	1T	2.5T
Long Context Tuning	Context	$32K_{\text{FoT}}$, $16K_{\text{no-FoT}}$	32K	32K	32K
	Batch Size	128_{FoT}, $16_{\text{no-FoT}}$	128	128_{FoT}, $32_{\text{no-FoT}}$	128
	Start Learning Rate	5e-5	1.5e-5	1.5e-5	1.5e-5
	End Learning Rate	5e-6	1.5e-6	1.5e-6	1.5e-6
	Warmup Steps	250	1000	1000_{FoT}, $200_{\text{no-FoT}}$	1000
	Tokens	1B	5.4B	2B	2B

Table 5.2: Training details. We pretrain a custom 270M parameter model and take a pretrained 3B/7B/ parameter OpenLLaMAv2 model (Geng, 2023) and 13B CodeLlama (Rozière et al., 2023) model. Subscript denotes that parameter was specific for a context extension method with FoT referring to (Tworkowski et al., 2023) and no-FoT to other methods (Naive, YaRN (Peng et al., 2023b), CodeLlama (Rozière et al., 2023)).

5.4.3 Experimental Results

Perplexity improvements

In Figure 5.2 we present perplexity improvements of 3B FoT SPLiCe BM25 over Baseline. Figure 5.3 shows the evolution of SPLiCe model perplexity during the training. We follow (Anthropic, 2023) and bucket perplexity by token positions in buckets of length 2^i up to 32768, and then average within the buckets. We average perplexity across arXiv, CUDA, Haskell, and CommonCrawl datasets.

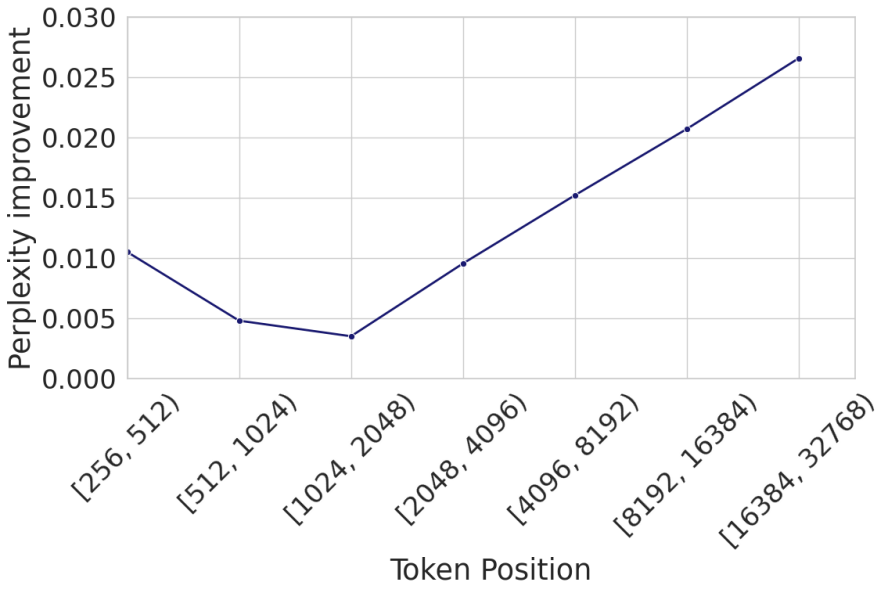


Figure 5.2: Perplexity improvement with SPLiCe against the Baseline of the final models (after 21k training steps). We bucket tokens by their positions in the document and calculate the average. Each dot is the difference of the averages of the SPLiCe and Baseline models. We observe that SPLiCe has smaller perplexity, and the improvements tend to be larger for tokens further in the document.

In-Context learning

To evaluate the improvements of in-context learning abilities, we use TREC (Li & Roth, 2002; Hovy et al., 2001) and DBpedia (Lehmann et al., 2015), which are text classification tasks. For TREC, to test the long context utilization, we vary the number of in-context examples in $\{780, 1560\}$ (which correspond roughly to $\{16, 32\}$K context length). We sample these examples multiple times and measure classification accuracy on the test set. In Table 5.3, we observe that SPLiCe improves the average performance across different context lengths and model sizes. We follow a very similar protocol for DBpedia ($\{16, 32\}$K context length corresponding to $\{190, 380\}$ in-context examples), confirming that SPLiCe improves the context utilization. In Appendix B.2, we study the distribution of improvements with respect to the choice of the in-context examples, showing the stochastic domination of SPLiCe over the baseline.

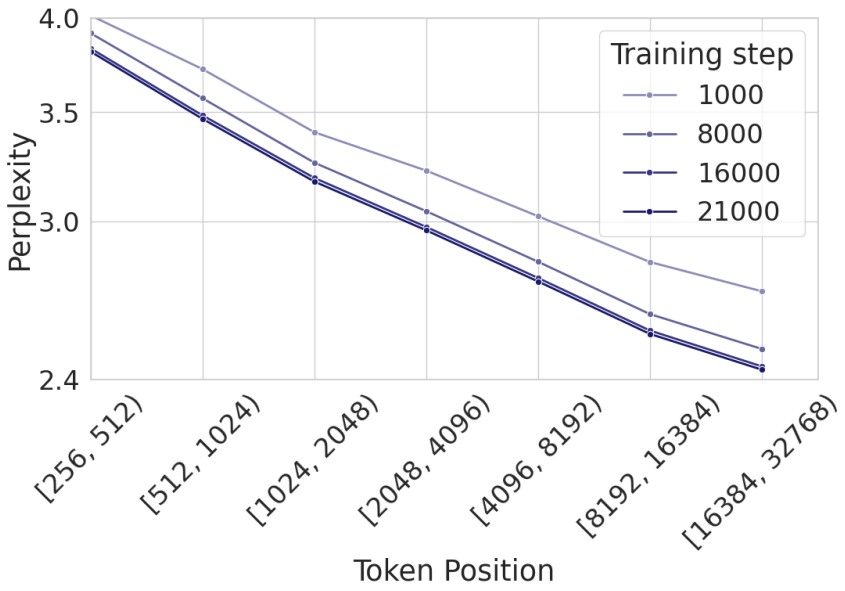


Figure 5.3: Evolution of the perplexity with SPLiCe, as the model is trained on more tokens. See Figure 5.2 for the difference with the baseline. As expected, SPLiCe significantly improves perplexity for tokens whose positions are very distant in the sequence. Perplexity for more distant tokens improves more significantly compared to tokens in the beginning, early in the training.

Task		**TREC**			**DBpedia**		
Model	Context	Baseline	SPLiCe	Δ [conf interv]	Baseline	SPLiCe	Δ [conf interv]
$3B_{FoT}$	32K	73.9	**79.3**	5.4 [4.7, 6.2]	82.9	**85.9**	3.0 [2.6, 3.6]
	16K	68.9	**76.9**	8.0 [6.9, 9.3]	79.1	**82.0**	2.9 [2.5, 3.4]
$7B_{FoT}$	32K	75.6	**79.4**	3.8 [2.1, 5.1]	82.9	**84.9**	2.0 [1.5, 2.4]
	16K	74.0	**79.0**	5.0 [3.4, 6.0]	83.6	**85.6**	2.0 [1.5, 2.5]
$7B_{CL}$	32K	75.3	**76.6**	1.3 [0.8, 1.8]	95.1	**95.6**	0.5 [0.3, 0.6]
	16K	81.4	**82.5**	1.1 [0.2, 1.6]	96.2	**96.4**	0.2 [0.0, 0.3]
$13B_{FoT}$	32K	89.2	**92.4**	3.2 [2.6, 3.8]	95.6	**96.0**	0.4 [0.0, 0.8]
	16K	88.2	**91.2**	3.0 [1.9, 3.5]	95.8	**96.8**	1.0 [0.6, 1.5]

Table 5.3: We test the average classification performance on TREC (Li & Roth, 2002; Hovy et al., 2001). We average across 50 sets of in-context examples for 3B models, 10 sets for 7B models, and 5 sets for 13B models. We use Δ [confidence interval] to denote the mean improvement and its 95% bootstrap confidence intervals (see Appendix B.2). For DBpedia, we average results across 40 sets of in-context examples for 3B and 7B models and 5 for 13B ones. Due to the size of the DBpedia evaluation dataset, for each set of in-context examples, we subsample a subset of 500 elements of the evaluation set.

Task	QASPER			HotPotQA			
Model	Context	Baseline	SPLiCe	Context	Baseline	SPLiCe	Δ [conf interv]
$3B_{FoT}$	32K	22.1	**22.8**	20K	**29.7**	29.6	-0.1 $[-0.3, 0.1]$
$7B_{FoT}$		22.8	**23.1**		26.0	**27.3**	1.3 $[1.2, 1.5]$
$7B_{CL}$		29.0	**29.7**		31.0	**31.2**	0.2 $[0.1, 0.6]$
$13B_{FoT}$		**32.0**	**32.0**		36.5	**36.7**	0.2 $[0.1, 0.4]$

Table 5.4: We measure question answering over long input using Qasper (Dasigi et al., 2021) (2-shot setting) and HotPotQA (Yang et al., 2018) (10-shot setting). For Qasper, we use the implementation from Language Model Evaluation Harness (Gao et al., 2021). For HotPotQA, we average results across 7 sets of in-context examples for 3B and 7B models and 2 for 13B ones, with Δ[confidence interval] denoting mean improvement and its 95% bootstrap confidence intervals. Note that in the 3B model case, despite using SPLiCe for code data only, we still have improvements in non-code tasks.

Question answering

We also show that SPLiCe models have improved question answering capabilities. To this end, we use Qasper (Dasigi et al., 2021) from SCROLLS (Shaham et al., 2022) and HotPotQA (Yang et al., 2018). Qasper contains questions regarding research papers provided as a part of the input context. In Table 5.4, we observe that SPLiCe improves the two-shot performance. Similarly, we observe improvements on HotPotQA, which tests the ability to aggregate knowledge across multiple documents. We stress that those results measure few-shot performance, as neither of the datasets was used during training.

Lost in the middle

The key retrieval task introduced in (Liu et al., 2023a) has become a routine diagnostic tool to study context capabilities. Specifically, the model is presented with a JSON-formatted list of key-value pairs, each being 128-bit UUIDs, and is tasked to retrieve the value for a given key. Even though very simple, the task proved to be challenging for many open-source language models. Specifically, LMs tend to achieve better performance when the relevant information is either in the beginning or the end of the input context, while struggling in the middle. The structure of the input is showcased below.

```
Extract the value corresponding to the specified key in the JSON object below.

JSON data:
{"4ef217b7-6bc0-48c6-af35-2765f1e730f3": "068192b7-16b1-40e0-8495-61c63f979d50",
 "cd6b8bdc-bc6c-4490-acb4-bc187a2dccba": "7364a26e-289f-4968-93d3-b273e882bdee",
 "7d057372-4ab8-4811-8110-658c3f19fff4": "3ad075c5-b567-4201-85a7-cb31a0c91540",
 "c62e192d-45e6-4646-bb88-1529c73256c9": "f0411644-1f6d-42a6-8af8-f06da66efc77",
 "06134e93-e158-490e-a66c-8e3b98e12735": "50a26a36-d832-450c-8d6e-a4cc3d0ec0ab",
 "3286f978-4270-4b54-8bfa-540d7e0772e6": "075cc716-1836-4f90-9be3-53e3d4ec6585",
 "4701aa05-c523-4b89-9700-64ab9c37c537": "49d86354-74c4-4256-9b3a-35e6e2b80d00",
 "c8895805-e574-4f13-9fe5-89da1d8c4748": "cc91af7f-8509-4bdc-bad7-2646af68e6d2"}
 "4701aa05-c523-4b89-9700-64ab9c37c537":
```

In Figure 5.4, we present the key retrieval performance of 7B models trained with CodeLlama (Rozière et al., 2023) context extension method. We note that despite relatively short tuning, SPLiCe significantly helps on hard-to-retrieve positions (the position of the key in the range $[25, 100]$)

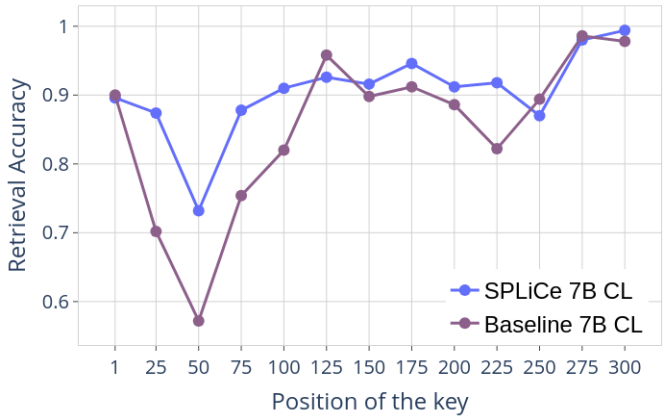
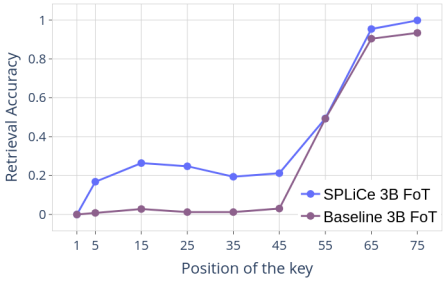
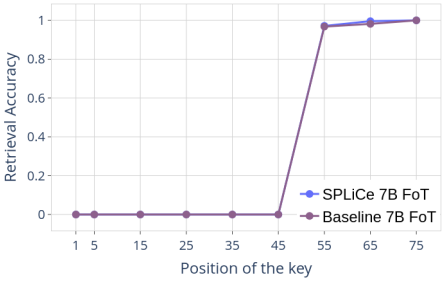


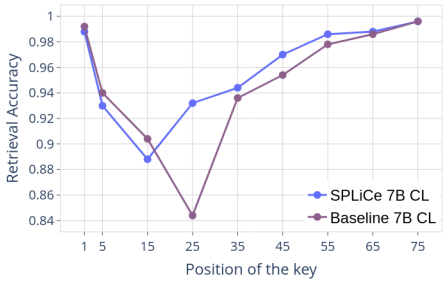
Figure 5.4: Key-value retrieval performance on a dictionary of 300 key-value pairs (≈ 24K tokens). The 7B CL model trained with SPLiCe achieves much higher accuracy on hard-to-retrieve positions in the middle. Each position was evaluated using 500 examples.



(a)



(b)



(c)

Figure 5.5: Performance on a smaller version of key-value retrieval task from (Liu et al., 2023a). We note that FoT models (a), (b) generally struggle to retrieve tokens that are only visible to a subset of layers with extended context. For comparison, we show the results with a model that has extended context in all layers (c) using CodeLlama (Rozière et al., 2023) method of context extension. Each position was evaluated using 500 examples.

We note that FoT-trained models struggle with this task. This is probably due to the fact that they extend context only in a couple of layers, and the key-value retrieval requires looking up and extracting a long sequence of letters and digits. Because of that, we evaluate FoT models with shorter dictionaries consisting of 75 key-value pairs (around 6K tokens) and show the results in Figures 5.5a, and 5.5b. For comparison, we also evaluate the 7B CL model with this context length and show the results in Figure 5.5c.

Ordering of examples

We typically use the identity ordering as `Order` in Algorithm 2 to merge documents into a single context, as we found that in most cases, it performs best. We found that random shuffling is slightly better in some cases. Specifically, this is the case of large 7B CodeLlama models; see Table 5.5. We hypothesize that random ordering forces the model to make use of the full space of the RoPe positional encoding. Whereas the identity ordering, also used in (Shi et al., 2023b), skews the model toward paying more attention to fragments of text that are not too far away. This suggests an interesting research direction on the intersection of data preparation and positional embeddings.

TREC			
Model	Context length (# of examples)	SPLiCe-no-shuf	SPLiCe-shuf
7B CL	32K (1560)	76.0 ±2.5	76.6 ±1.9
	8K (380)	77.1 ±2.4	76.5 ±1.8

Table 5.5: Average classification performance on TREC (Li & Roth, 2002; Hovy et al., 2001). We compare 7B CL model trained on SPLiCe prepared data with different approaches to ordering the examples (different function `Order` in 2). SPLiCe-shuf denotes the model trained on data that shuffled the documents randomly in context, for SPLiCe-no-shuf `Order` was the identity function. We use ± to denote the standard deviation. We decided to stick with the model trained with random shuffling as it has slightly better long-context performance and lower standard deviation.

SPLiCe parameters

There are two important design choices related to SPLiCe. First, how many related documents are retrieved in each step (the parameter k in Algorithm 2). Second, how the documents are ordered. Table 5.6 indicates that $k = 1$ is the best choice, though the differences are rather small. We found that changing the order of documents in training examples hardly matters for 270M models. We use 'standard', as ordered by Algorithm 2, the reversed order, and random shuffling.

Short context evaluation

An undesirable side effect of fine-tuning for extended context (e.g., with SPLiCe) could be performance degradation on shorter contexts. To test this, we evaluate the short context performance of our models using GSM8K (Cobbe et al., 2021), MMLU (Hendrycks et al., 2021) and TREC (Li & Roth, 2002; Hovy et al., 2001). We present the results in Table 5.7. We note that for smaller models, both SPLiCe and Baseline show some

Method	Top-k	Order	Code				
			C++	Haskell	Python	CUDA	All
SPLiCe BM25	top 1	standard	**2.38**	**3.20**	**2.82**	**2.23**	**2.93**
		reverse	**2.38**	3.21	**2.82**	**2.23**	**2.93**
	top 2	standard	2.39	3.23	2.84	2.24	2.95
		reverse	2.39	3.24	2.84	2.24	2.95
	top 3	standard	2.39	3.24	2.84	2.25	2.97
		reverse	2.39	3.25	2.84	2.25	2.96
		shuffle	2.40	3.24	2.84	2.25	2.96

Table 5.6: Ablation of SPLiCe hyper-parameters. For each ablation, we have trained the same $270M$ parameter model using different data organization methods. Top-k corresponds to the number of descendants chosen in the `RETRIEVE`(d, k) step of the Algorithm 2. Reverse and shuffle correspond to the final order of examples C returned by the Algorithm 2 (reverse – the order of documents in C is reversed, shuffle – examples are shuffled.

short context degradation, whereas the larger 13B CodeLlama Model SPLiCe brings significant improvements in short context performance on GSM8K (+1.7), TREC (+3.3) and maintains the MMLU performance.

Model	MMLU					GSM8K	TREC
	STEM	Humanities	Social	Other	Mean		
3B Starting Chkpt	24.4	25.6	26.3	27.8	25.9	4.3	58.4
$3B_{FoT}$ SPLiCe	25.8	26.7	26.3	27.3	26.5	2.5	66.4
$3B_{FoT}$ Baseline	25.2	26.2	24.7	26.9	25.7	3.4	61.3
7B Starting Chkpt	33.6	42.1	46.0	44.5	40.8	8.0	59.6
7B 2K Tuned	33.7	40.8	44.6	41.1	39.4	8.4	62.3
$7B_{FoT}$ SPLiCe	31.0	35.6	40.3	40.3	36.3	7.6	63.7
$7B_{FoT}$ Baseline	30.1	36.8	40.3	39.8	36.2	6.7	62.9
$7B_{CL}$ SPLiCe	32.7	38.8	37.8	38.0	36.5	5.9	59.1
$7B_{CL}$ Baseline	32.7	37.5	38.1	37.5	36.1	6.3	59.1
13B Starting Chkpt	36.6	**48.2**	**50.6**	**45.4**	**44.3**	21.4	78.7
$13B_{FoT}$ SPLiCe	**38.3**	**48.6**	48.7	44.5	**44.4**	**23.1**	**82.0**
$13B_{FoT}$ Baseline	**39.0**	47.5	48.8	**45.7**	**44.7**	21.9	77.6

Table 5.7: We evaluate our models on MMLU (5-shot), GSM8K (8-shot) and TREC (90-shot). We provide an additional comparison with their starting checkpoint. For 7B case, we additionally compare with a model tuned with 2k context length on the same data. For each task, we highlight best results up to 1 point.

5.5 Detailed Study with Medium Models

In Table 5.8 and Table 5.9, we present a comprehensive examination of the impact of document packing on long-context performance using 270M parameter models. Using the smaller size allows us to train over 30 models and, thus, more accurately estimate the impact of various design choices.

5.5.1 Training Protocol

Initially, we train with the 2K context length on 6.3B tokens from RedPajama (TogetherComputer, 2023). Subsequently, we continue training using 1B tokens with the context extended to 32K on a mixture of the original RedPajama data (TogetherComputer, 2023) and long-context data created using SPLiCe/Baseline. The amount of pre-training tokens is based on scaling laws from (Hoffmann et al., 2022) and constants calculated for GPT-like models in (Karpathy, 2022).

We employ the Focused Transformer (FoT) objective (Tworkowski et al., 2023) for context extension (unless stated otherwise). This approach is motivated by practical factors, such as training with a short context length, which expedites the process, while context scaling can be achieved by fine-tuning on a relatively small amount of tokens, as demonstrated by (Chen et al., 2023; Tworkowski et al., 2023). Loosely inspired by (Ouyang et al., 2022; Rozière et al., 2023), in the latter phase, long context data (i.e., prepared with SPLiCe) constitutes half of the mixture. We also examine how SPLiCe works with other context extension methods in Section 5.5.3.

5.5.2 Evaluation

We measure perplexity on held-out portions of the arXiv (Azerbayev et al., 2022) and StarCoder (Li et al., 2023b) datasets, employing a context length of 32K. The selection of these datasets is motivated by the fact that they can benefit from long-context information, as demonstrated in (Chen et al., 2023; Li et al., 2023b). For example, functions are often re-utilized, and similar terms are employed across papers. We exclude documents with fewer than 32K tokens and truncate those exceeding this length. In Appendix B.1, we evaluate our models using short context data, confirming no performance degradation with respect to the base model.

5.5.3 Data

Evaluation Data

We have taken a random subset of arXiv from Proof-pile. For StarCoder data, we have downloaded up to 64GB of each of the mentioned language subsets and performed a random 85/15 split for the languages we train on.

When evaluating the perplexity of the model, we skip documents shorter than the model context and truncate documents longer than it.

Training Data

The StackExchange data was taken from the Proof-pile. To prepare the code train data, we take the StarCoder train splits mentioned in Section 5.5.3, shuffle them, group the documents by the repository (documents from the same repository occur one after another), and split them into smaller packs. We also split repos larger than 25MB and filter out files that are longer than 30k characters. The reason behind repo splitting is to avoid the situation where one repository occupies a significant portion of the data pack. We have noticed repos containing as many as 40K files and files as long as 11M characters. The character filtering is consistent with our method as we aim to improve the performance in a scenario that lacks high-quality long-context data. For C# and Python, only one pack is used to organize the data. For C, we have performed a run on three packs and provided results and standard deviations in Table 5.9. For large models,

we run the methods on several packs and concatenate the results into a single dataset. For natural language datasets, we extract a random subset of documents.

Altered Data	Method	arXiv	Code				Code & arXiv
			Haskell	Python	CUDA	All	
C	SPLiCe BM25	**5.46** .09	**3.20** .17	**2.81** .12	**2.22** .11	**2.94** .13	**3.10** .13
	SPLiCe Cont	5.48 .07	3.22 .15	2.82 .11	2.23 .10	2.96 .11	3.11 .12
	SPLiCe Repo	5.47 .08	3.22 .15	2.83 .10	2.24 .09	2.96 .11	3.12 .11
	Baseline	5.55	3.37	2.93	2.33	3.07	3.23
C#	SPLiCe BM25	**5.52 .13**	**3.33 .25**	**2.90 .17**	**2.46 .19**	**3.11 .20**	**3.26 .20**
	SPLiCe Cont	5.53.12	3.35.23	2.91.16	2.48.17	3.12.19	3.27.19
	SPLiCe Repo	5.53 .12	3.35 .23	2.91 .16	2.49 .16	3.12 .19	3.27 .19
	Baseline	5.65	3.58	3.07	2.65	3.31	3.46
Python	SPLiCe BM25	**5.47** .10	**3.25** .21	**2.53** .09	**2.41** .15	**3.02** .15	**3.17** .15
	SPLiCe Cont	5.49 .08	3.28 .18	**2.53** .09	2.43 .13	3.03 .14	3.19 .13
	SPLiCe Repo	5.48 .09	3.27 .19	2.54 .08	2.44 .12	3.03 .14	3.18 .14
	Baseline	5.57	3.46	2.62	2.56	3.17	3.32
Wikipedia	SPLiCe BM25	**5.64** .09	**3.82** .15	**3.26** .11	**2.87** .13	**3.55** .13	**3.68** .13
	SPLiCe Cont	5.65 .08	3.87 .10	3.30 .07	2.92 .08	3.59 .09	3.72 .09
	Baseline	5.73	3.97	3.37	3.00	3.68	3.81
StackEX	SPLiCe BM25	**5.07** .07	**3.88** .06	**3.32** .04	**2.89** .05	**3.60** .05	**3.69** .05
	SPLiCe Cont	5.09 .05	3.91 .03	3.35 .01	2.93 .01	3.63 .02	3.73 .01
	Baseline	5.14	3.94	3.36	2.94	3.65	3.74

Table 5.8: Perplexity with an improvement over Baseline highlighted in the subscript: (improvement over Baseline). We fine-tune a 270M parameter model with a 32K context on a 50/50 mixture of RedPajama (organized in a standard way) and long-context data: C, C#, Python, Wikipedia, StackExchange, prepared using a method of choice (SPLiCe BM25, SPLiCe Cont, SPLiCe Repo, Baseline). Baseline denotes organizing long-context data in the same way as RedPajama. SPLiCe beats the Baseline often by a large margin. The variants of SPLiCe perform similarly, with SPLiCe BM25 being slightly better. For detailed results, see Appendix B.3.

Long Context Data	Method	arXiv	Code		Code & arXiv
			Python	All	
C	SPLiCe BM25	**5.463** ± .002	**2.810** ± .002	**2.942** ± .005	**3.100** ± .004
	SPLiCe Cont	5.477 ± .005	2.824 ±.001	2.957 ± .006	3.115 ± .006
	SPLiCe Repo	5.474 ± .007	2.827 ± .006	2.958 ± .009	3.115 ± .009
	Baseline	5.550 ± .002	2.931 ± .008	3.073 ± .006	3.228 ± .005

Table 5.9: Perplexity fine-tune on a 50/50 data mixture of RedPajama and C code. We report the mean and standard deviation. Interestingly, training on the code data with SPLiCe improves general long-context performance on arXiv.

Different Context Extension Methods

We test SPLiCe with different context extension methods. In Table 5.10, we show that SPLiCe brings improvements also when context is extended in all layers using the

naive approach, the CodeLlama (Rozière et al., 2023), and YaRN (Peng et al., 2023b) adjustment methods.

RoPe scale	Method	arXiv	Code				Code & arXiv
			Haskell	Python	CUDA	All	
Naive	SPLiCe BM25	**6.25** .08	**4.84** .19	**3.55** .11	**2.84** .12	**3.72** .13	**3.88** .12
	SPLiCe Repo	**6.25** .08	4.87 .16	3.56 .10	2.85 .11	3.74 .11	3.89 .11
	Baseline	6.33	5.03	3.66	2.96	3.85	4.00
CL	SPLiCe BM25	**5.74** .02	**4.28** .09	**3.22** .05	**2.53** .05	**3.34** .06	**3.49** .06
	SPLiCe Repo	**5.74** .02	**4.28** .09	**3.22** .05	2.54 .04	3.35 .05	3.50 .05
	Baseline	5.76	4.37	3.27	2.58	3.40	3.55
YaRN	SPLiCe BM25	**5.77** .02	**4.32** .10	**3.24** .05	**2.55** .06	**3.37** .07	**3.52** .06
	SPLiCe Repo	**5.77** .02	**4.32** .10	**3.24** .05	2.56 .05	3.38 .06	3.53 .05
	Baseline	5.79	4.42	3.29	2.61	3.44	3.58

Table 5.10: Perplexity (improvement over Baseline) for training on a 50/50 data mixture of RedPajama and C#. We check that SPLiCe brings improvements when fine-tuning for the longer context (16K) using the method of CodeLlama (Rozière et al., 2023), YaRN (Peng et al., 2023b), or left without changes (Naive), as opposed to FoT used in the other experiments. For details, see Table B.5.

5.5.4 Summary of the Results

Structuring data improves performance. We observe that all tested variants of SPLiCe outperform Baseline, often significantly, regardless of the training and evaluation dataset. This indicates that the structure of the data is important for long-context training.

The positive effects transfer between domains. We observe (see Tables 5.4 and 5.8) that the positive effects of SPLiCe transfer between domains when training on code and evaluating on Qasper/arXiv, and when training on web data and evaluating on code. This is a very interesting effect, indicating that training induces generic long-context capabilities. Following that, we conjecture that better data mixtures and synthetic long-context data could bring further benefits.

SPLiCe improves the performance of various context extension methods. SPLiCe only changes the input structure, and conjecturally, it should work with any context extension method. Our main experiments use the Focused Transformer (FoT) approach (Tworkowski et al., 2023). FoT extends the context in only a couple of attention layers and does not change the positional embeddings of the remaining layers. In Table 5.10 in Section 5.5.3, we confirm that SPLiCe integrates seamlessly with methods that extend the context in all layers. Specifically, we test adjusting Rotary Positional Encodings according to either the recipe as in CodeLlama (Rozière et al., 2023), YaRN (Peng et al., 2023b), or leaving them unchanged.

Training is stable. We prepared three subsets of the C code dataset and tested the methods on each of them. In Table 5.9, we report the mean perplexity and its standard deviation. We observe that the differences between subsets are minimal, indicating training stability and confirming the statistical significance of our results.

5.6 Limitations and Future Work

We demonstrate that structuring the training data is a viable way to improve the model's performance. The presented method, SPLiCe, can be viewed as a general framework for organizing documents into training examples. This opens multiple further research avenues.

Degree of relevance. SPLiCe constructed training examples by concatenating the most relevant documents (according to BM25/Cont/Repo). However, recent results by Tworkowski et al. (2023) show that introducing unrelated data into the context can help the model learn better representations. We leave the study of how the choice of retriever (in particular, the ratio of related and unrelated documents) affects performance for future work.

In order to prepare the training data, SPLiCe uses each document exactly once (we mask out each used document for further retrieval). However, it is possible that allowing some high-quality documents to occur more than once may be beneficial.

Retrieval granularity. Another avenue for future work is to study the granularity of the pieces from which the training examples are constructed. In this chapter, we focused on the document-level granularity. However, it is possible to construct training examples from smaller pieces.

Other context extension methods. In most cases, our models were trained for a long context using the methodology presented in (Tworkowski et al., 2023). Additionally, we tested three popular context extension methods on a medium scale (Naive, YaRN, and CodeLlama) and tuned a large 7B model using the CodeLlama approach, preliminarily confirming the applicability of SPLiCe. However, we leave more detailed studies with other context extension methods to future work.

Other data sources. One approach to training Long-Context Language Models (LCLMs) is to use conversational data (Li et al., 2023a). This is complementary to our method. SPLiCe can utilize data that already exists in vast quantities and can be easily applied to different types of text (such as code, Wikipedia articles, or StackExchange questions and answers) to further increase the number of long-context examples. We leave researching how SPLiCe integrates with other methods for preparing long-context data as future work.

Data curation. Using highly correlated samples has the potential to result in training instability. However, we observed no performance degradation during our experiments. We leave the study of how SPLiCe integrates with different data types for the future.

5.7 Conclusions

In this chapter, we presented SPLiCe, a method for constructing training examples for Long-Context Language Models (LCLMs). It utilizes BM25/Contriever-MSMARCO to find relevant documents and feed them to the model in a structured manner. We show that SPLiCe improves performance on downstream tasks and the language modeling abilities of LLMs. We further show that SPLiCe can be used to improve long-context utilization of large-scale models using only short fine-tuning. We believe the presented results indicate multiple interesting research directions for improving the performance of LCLMs with structured data.

Chapter 6

MoE-Mamba: Efficient Selective State Space Models with Mixture of Experts

> "Now the serpent was more crafty than any of the wild animals
> the LORD God had made."
> — Genesis 3:1, *New International Version*

6.1 Introduction

Recently, a new architecture for language modeling has attracted significant attention in the research community. Specifically designed for efficient long context, Mamba was introduced by Gu & Dao (2023). This architecture, based on structured state space models (SSMs), promises to improve with scale similarly to the Transformer architecture while avoiding the quadratic complexity of the attention layer with respect to the number of tokens in the context. While techniques like LSH attention, used in Chapter 2, and memory attention, used in Chapter 5, aim to mitigate this problem, further improvements in SSMs or similar techniques could potentially replace the Transformer as the primary architecture for LLMs.

The preceding chapters described techniques embedded in or using the Transformer architecture. Thus, questions arise: Do techniques improved in this work, such as the Mixture of Experts, depend on the Transformer architecture? Or could the research community still benefit from MoE if an architecture better than the Transformer was found?

In this chapter, we advocate that to unlock the potential of any architecture, including SSMs, they should be combined with Mixture of Experts. To this end, we introduce **MoE-Mamba**—a model that combines Mamba (Gu & Dao, 2023) with a Switch layer (Fedus et al., 2022). MoE-Mamba enables the efficiency gains of both SSMs and MoE, outperforming both Mamba and Transformer-MoE. Through comprehensive studies, we confirm that the effect is robust to design choices and the number of experts. Our results indicate a very promising research direction that may allow scaling SSMs beyond tens of billions of parameters and competing with the largest state-of-the-art language models.

6.2 Related Work

State Space Models and Related Attention-Free Architectures. State Space Models (SSMs) (Gu et al., 2022b, 2021, 2022a; Gupta et al., 2022; Li et al., 2022; Ma et al., 2022; Orvieto et al., 2023; Smith et al., 2023) constitute a family of architectures used for sequence modeling. Originating from signal processing, these models can be seen as a combination of RNNs and CNNs (Gu & Dao, 2023). Although they potentially offer considerable benefits, a number of issues have been identified with SSMs (Gu et al., 2022b), preventing them from becoming the leading architecture for language modeling. However, recent breakthroughs (Gu et al., 2022b; Fu et al., 2023; Smith et al., 2023; Gu & Dao, 2023) have enabled deep SSMs to be increasingly competitive against Transformers (Vaswani et al., 2017). In addition to SSMs, numerous other architectures that do not rely on the quadratic attention mechanism have been proposed (Zhai et al., 2021; Poli et al., 2023; Sun et al., 2023; Peng et al., 2023a).

Mamba. Mamba (Gu & Dao, 2023) is a recently introduced SSM-based model that achieves remarkable, Transformer-like performance. By employing a work-efficient parallel scan, Mamba mitigates the impact of the sequential nature of recurrence, while fusing GPU operations removes the requirement to materialize the expanded state. Intermediate states required for backpropagation are not saved but recomputed during the backward pass, thereby reducing memory requirements. The advantages of Mamba over the attention mechanism are especially prominent during inference, as not only is the computational complexity reduced, but memory usage is also independent of context length. Figure 6.5 shows the inner structure of a Mamba layer in Section 6.5, which explores integrating MoE directly into the Mamba layer.

Mamba Combined with MoE. Subsequent to the publication of MoE-Mamba, Anthony et al. (2024) presented an architecture similar to MoE-Mamba, demonstrating the potential of connecting Mamba with MoE on downstream tasks, which validates our findings. In contrast to their work, we run extensive ablations on the model architecture, number of experts, and other design choices. We also explore the potential of integrating Conditional Computation into the Mamba block. Later, Lieber et al. (2024) introduced a heterogeneous architecture scaled to 52B parameters, interleaving Mixture of Experts, feed-forward, Mamba, and attention layers.

6.3 MoE-Mamba

In this section, we present the architectural details of our model, MoE-Mamba, while Sections 6.5.1 and 6.5.2 explore its variants and related approaches.

While the design of a single Mamba block is complex, the overall architecture is quite simple: it consists of multiple Mamba blocks stacked one after another, with each layer's output being added to the residual stream (see Figure 6.1). This design is extremely similar to the Transformer architecture, with both feed-forward and attention blocks replaced by Mamba (see comparison in Figure 6.1). Mamba also uses the same design for both embedding and unembedding layers. By design, Mamba lacks positional embedding added to token embedding, which is present in traditional Transformers. However, in modern Transformers, relative positional embeddings such as RoPE (Su et al., 2021) are integrated into the attention block itself.

In MoE-Mamba, we interleave Mamba layers with MoE layers (see Figure 6.1), utilizing the design of MoE introduced in Switch Transformer Fedus et al. (2022).

This way, MoE-Mamba separates unconditional processing of every token by the Mamba layer—which can efficiently integrate the whole sequence context into an internal representation—and conditional processing by an MoE layer that can apply the most relevant expert (and thus the subset of parameters) for each token. The idea of interleaving conditional and unconditional processing is used in some MoE-based models, typically by alternating vanilla and MoE feed-forward layers (Lepikhin et al., 2020; Fedus et al., 2022).

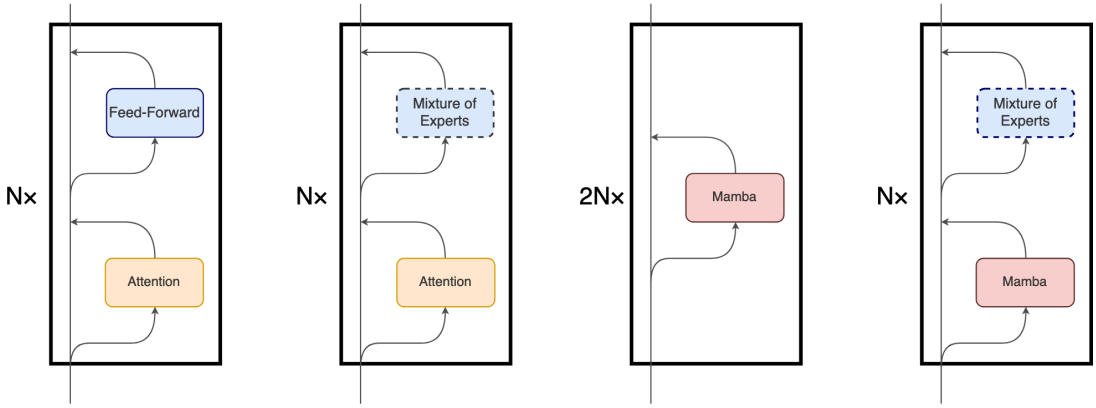


Figure 6.1: Diagrams of the architectures. From the left: Transformer, Transformer-MoE, Mamba, MoE-Mamba.

6.4 Experiments

In this section, we provide empirical validation of our hypothesis that interleaving Mamba with MoE can improve the performance of a model. We compare MoE-Mamba to three baselines: Mamba, Transformer, and Transformer-MoE. All models in our experiments are decoder-only.

In the standard Transformer architecture, a single attention layer contains $4d^2_{\text{model}}$ parameters, whereas a feed-forward layer contains $8d^2_{\text{model}}$ parameters. A single Mamba layer contains slightly over $6d^2_{\text{model}}$ parameters (Gu & Dao, 2023).

To compare MoE-Mamba to Transformer and Mamba baselines, we scale down the size of each expert in our model (setting $d_{\text{expert}} = 3d_{\text{model}}$). This way, we keep both the number of blocks and the number of active parameters per token roughly the same across all models of similar size. Active parameters denote those used to calculate the output for a given token (e.g., typically, only one expert in each MoE layer is active). In this chapter, we report the number of active parameters (excluding embedding and unembedding layers) and not the number of floating-point operations (FLOPs), following Zhou et al. (2022). Both numbers will be roughly proportional (Kaplan et al., 2020), but the number of FLOPs is harder to calculate and less relevant for hardware-aware architectures like Mamba with its optimizations, especially during inference.

For all models and their variants, we report the number of trainable, non-embedding parameters, i.e., we exclude the parameters in the input (embedding) and output (unembedding) layers. This convention is proposed by Kaplan et al. (2020), who note that using just non-embedding parameters gives their scaling laws a clearer form. The relatively low importance of the number of embedding parameters for the final performance has been noted by Lan et al. (2020).

6.4.1 Training Setup

Due to computational constraints, we perform most of our experiments on smaller, $\square_{25M}$ models and validate our findings on $\square_{100M}$ models.

We train the models on the C4 dataset (Raffel et al., 2020) with the standard language modeling task of next-token prediction. We optimize and report cross-entropy loss, which equals log perplexity per token.

All models use the GPT-2 tokenizer (Radford et al., 2019). We train the models using PyTorch (Paszke et al., 2019) and utilize FSDP (Zhao et al., 2023) to facilitate a multi-GPU setup.

6.4.2 Model and Training Hyperparameters

In this chapter, as in Chapter 3, basic model hyperparameters (d_{model}, d_{ff}, the number of attention heads, the number of layers) were inspired by BERT Devlin et al. (2018); Turc et al. (2019), with the $\square_{25M}$ models being equivalent to $\text{BERT}_{\text{MEDIUM}}$, and $\square_{100M}$ models copying $\text{BERT}_{\text{BASE}}$ configuration while increasing the number of blocks from 12 to 16. The learning rate schedule, as well as weight decay, and gradient clipping values were set according to community's standard practices. We used the AdamW optimizer Loshchilov & Hutter (2019). We tune the learning rate separately for all $\square_{25M}$ models and reuse it when training their $\square_{100M}$ counterparts. When training Transformer-MoE_{100M}, we halve the learning rate due to instabilities.

The main experiments, described in Section 6.4.3, use approximately 10B tokens for $\square_{25M}$ models and around 30B tokens for $\square_{100M}$ models. The experiments described in Section 6.5 use 1B tokens.

To further encourage an even distribution of tokens to experts, load-balancing loss, as described by Fedus et al. (2022), with weight $\alpha = 0.01$ was added to the training objective. We use a capacity factor of 1.0 to fairly compare the number of training FLOPs between architectures.

Hyperparameter		**Transformer$_{25M}$**	**Mamba$_{25M}$**	**Transformer-MoE$_{25M}$**	**MoE-Mamba$_{25M}$**
Model	Total Blocks	8	16	8	8
	d_{model}	512	512	512	512
	# Parameters	25M	27M	545M	542M
	# Active Parameters per Token	25M	27M	25M	26M
feed-forward	d_{ff}	2048	-	-	-
Mixture of Experts	d_{expert}	-	-	2048	1536
	N_{experts}	-	-	32	42
Position Embedding		RoPE	-	RoPE	-
Attention	N_{heads}	8	-	8	-
Training	Training Steps	150K	150K	150K	150K
	Context Length	1024	1024	1024	1024
	Batch Size	64	64	64	64
	Max Learning Rate	5e-4	1e-3	5e-4	5e-4
	LR Warmup	1%	1%	1%	1%
	LR Schedule	Cosine	Cosine	Cosine	Cosine
	Final LR Ratio	0.1	0.1	0.1	0.1
	Weight Decay	0.1	0.1	0.1	0.1
	Gradient Clipping	0.5	0.5	0.5	0.5

Table 6.1: Hyperparameters for $\square_{25M}$ models.

Hyperparameter		Mamba_{100M}	$\text{Transformer-MoE}_{100M}$	MoE-Mamba_{100M}
Model	Total Blocks	32	16	16
	d_{model}	768	768	768
	# Parameters	121M	2454M	2439M
	# Active Parameters per Token	121M	114M	117M
Mixture of Experts	d_{expert}	-	3072	2304
	N_{experts}	-	32	42
Position Embedding		-	RoPE	-
Attention	N_{heads}	-	12	-
Training	Training Steps	30K	30K	30K
	Context Length	1024	1024	1024
	Batch Size	1024	1024	1024
	Max Learning Rate	1e-3	2.5e-4	5e-4
	LR Warmup	1%	1%	1%
	LR Schedule	Cosine	Cosine	Cosine
	Final LR Ratio	0.1	0.1	0.1
	Weight Decay	0.1	0.1	0.1
	Gradient Clipping	0.5	0.5	0.5

Table 6.2: Hyperparameters for $\square_{100M}$ models.

6.4.3 Comparing Architectures

Table 6.3 compares the training results of MoE-Mamba and baselines; see also Figure 6.2 for log perplexity curves. MoE-Mamba shows a remarkable improvement over the vanilla Mamba model across both model sizes. Notably, MoE-Mamba_{100M} performed on par with the vanilla Mamba_{100M} with a 2.35× speedup in terms of processed tokens. For the $\square_{25M}$ model size, these performance gains are lower, likely due to a smaller number of training tokens. More generally, we observe that the gains increase over the training, oscillating around $1.6\times - 1.9\times$ for $\square_{25M}$ models after the initial training period. Further discussion of the speedup can be found in Section 6.6.1. We observe that MoE-Mamba performs better than the corresponding Transformer-MoE, which supports the findings by Gu & Dao (2023) that Mamba is a competitive alternative to the Transformer.

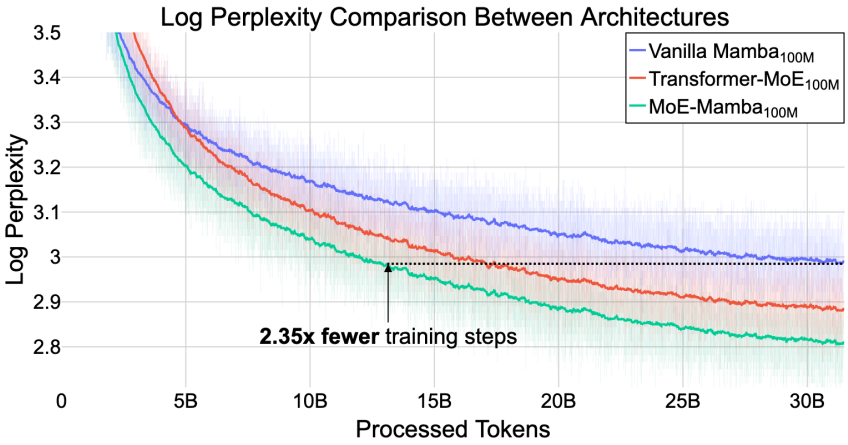


Figure 6.2: Log perplexity throughout the training. From top to bottom: Mamba_{100M}; $\text{Transformer-MoE}_{100M}$; MoE-Mamba_{100M}.

We use EMA-smoothed ($\alpha = 0.001$) training log perplexity as the comparison metric for both final loss and speedup measurements since it is a more fine-grained comparison metric than test log perplexity. The test log perplexity comparison for $\square_{100M}$ models can be found in Section 6.6.3.

Model	**#Parameters**	**#Active Parameters per Token**	**Final Log Perplexity**	**Speedup Over Vanilla Mamba (Training Steps)**
Mamba_{25M}	27M	27M	3.34	1
MoE-Mamba_{25M} (ours)	542M	26M	**3.19**	**1.76**
$\text{Transformer-MoE}_{25M}$	545M	25M	3.23	1.56
Transformer_{25M}	25M	25M	3.43	>1
Mamba_{100M}	121M	121M	2.99	1
MoE-Mamba_{100M} (ours)	2439M	117M	**2.81**	**2.35**
$\text{Transformer-MoE}_{100M}$	2454M	114M	2.88	1.79

Table 6.3: Comparison between different architectures. The $\square_{25M}$ models were trained on approximately 10B tokens, and the $\square_{100M}$ models were trained on approximately 30B tokens. The numbers of total and active parameters are not matched exactly between similarly-sized models due to reasons such as the MoE models including routers and the Mamba layer not containing precisely $6d^2_{\text{model}}$ parameters—a design choice we did not want to modify. We consider those differences to be too small to significantly affect our results.

6.4.4 Number of Experts

Number of Experts	**#Parameters**	**#Active Parameters per Token**	**Log Perplexity After 1B Tokens**	**Speedup Over Vanilla Mamba (Training Steps)**
N/A - Vanilla Mamba	27M	27M	3.72	1
1	26M	26M	3.75	<1
4 experts	64M	26M	3.72	1.03
8 experts	114M	26M	3.70	1.10
16 experts	215M	26M	3.67	1.21
32 experts	416M	26M	**3.67**	**1.23**

Table 6.4: Log perplexity after 1B tokens for various numbers of experts. Note that the parameter counts exclude the embedding and unembedding layers.

Table 6.4 and Figure 6.3 show the training runs for different numbers of experts. The results indicate that our approach scales favorably with the number of experts. MoE-Mamba outperforms vanilla Mamba when $N_{\text{experts}} \geq 4$. We obtain the best result with 32 experts and expect further gains with even more experts.

Interestingly, models with a small number of experts perform worse than vanilla Mamba. This is consistent with Gu & Dao (2023), reporting that Mamba interleaved with feed-forward layers (which corresponds to an MoE layer with a single expert) is worse than vanilla Mamba.

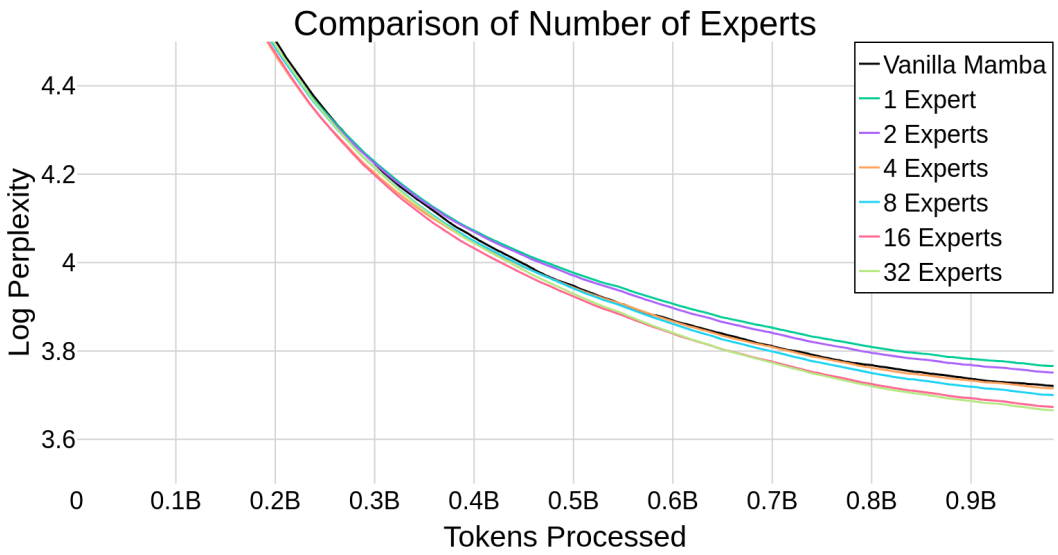


Figure 6.3: Smoothed training loss (log perplexity) for a differing number of experts for MoE-Mamba with approximately 26M active non-embedding parameters. The final log perplexity improves monotonically as the number of experts increases.

6.4.5 Optimal Ratio of Active Parameters in Mamba and MoE

The allocation of FLOPs and parameters to various components is an important design choice in heterogeneous architectures. For example, in the Transformer, the model's structure has been studied extensively by Kaplan et al. (2020). Here, we examine the optimal ratio of active parameters in the Mamba layer to the number of active parameters in the MoE layer.

In this section, we investigate the optimal ratio of active parameters in the Mamba layer to active parameters in the MoE layer while keeping the total number of parameters fixed. Under these constraints, a given ratio determines the so-called expansion factor, E, of the Mamba layer, the number of experts, and their sizes, as detailed in Table 6.5 (see also Figure 6.5 for Mamba design).

The results are presented in Figure 6.4. We observe that increasing the number of active Mamba parameters improves performance. However, the gains become marginal after reaching the $3:3$ ratio, and higher ratios are impractical due to inefficient hardware utilization and high routing costs caused by a large number of experts. We default to this choice in all other experiments.

Figure 6.4 may initially suggest that increasing the ratio enhances performance, and perhaps assigning all the active parameters to Mamba would yield the best performance (ratio *"6:0"*). However, it should be noted that all the investigated models contain the same number of total parameters and active parameters per token. A hypothetical model described above could not achieve this property. If we loosen the requirement and place all the parameters in Mamba, the resulting model is the same as $\text{Mamba}_{25\text{M}}$ with the expansion factor $E = 4$ and 8 instead of 16 Mamba layers. This model achieves marginally worse final log perplexity than $\text{Mamba}_{25\text{M}}$ (3.73).

Ratio $N_{\text{Mamba}}^{\text{act. params}} : N_{\text{MoE}}^{\text{act. params}}$	Expansion Factor E (Mamba)	Expert Size	Number of Experts
$1:5$	$\frac{2}{3}$	2560	19
$2:4$	$1\frac{2}{3}$	2048	24
$3:3$	2	1536	32
$4:2$	$2\frac{2}{3}$	1024	48
$5:1$	$3\frac{1}{3}$	512	96

Table 6.5: Comparison of different ratios of parameters between Mamba and MoE. The $E = 2$ corresponds to MoE-Mamba$_{25\text{M}}$. The total number of parameters in all models is 542M, and the number of active parameters per token is 26M.

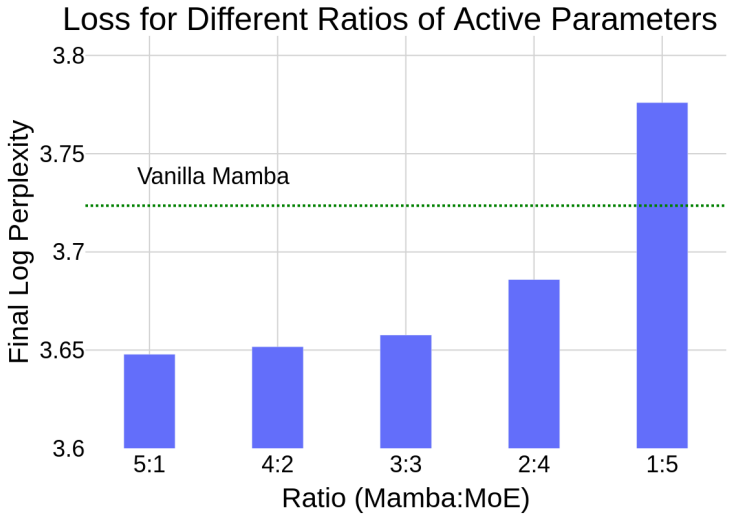


Figure 6.4: Final log perplexity at different ratios of active Mamba-to-MoE active parameters. Note that the MoE layer contains the majority of the total parameters in each model.

6.5 Alternative Designs

6.5.1 Parallel MoE-Mamba

Inspired by Wang (2021) and Chowdhery et al. (2022), we experiment with an alternative block design where the MoE feed-forward layer and the Mamba layer are placed in parallel, rather than sequentially (see Figure 6.5). We compare this design to MoE-Mamba across various numbers of experts (see Figure 6.6). MoE-Mamba consistently outperforms this variant in all tested settings. The parallel MoE-Mamba matches vanilla Mamba when $N_{\text{experts}} \geq 8$, but it requires between 2 and 4 times as many experts and total parameters to match the performance of the sequential variant.

While it may be an attractive alternative in some cases—for example, due to potentially enabling more efficient use of hardware through different communication strategies (Wang, 2021) or fused input matrix multiplications (Chowdhery et al., 2022)—in general, the sequential MoE-Mamba would be the recommended choice.

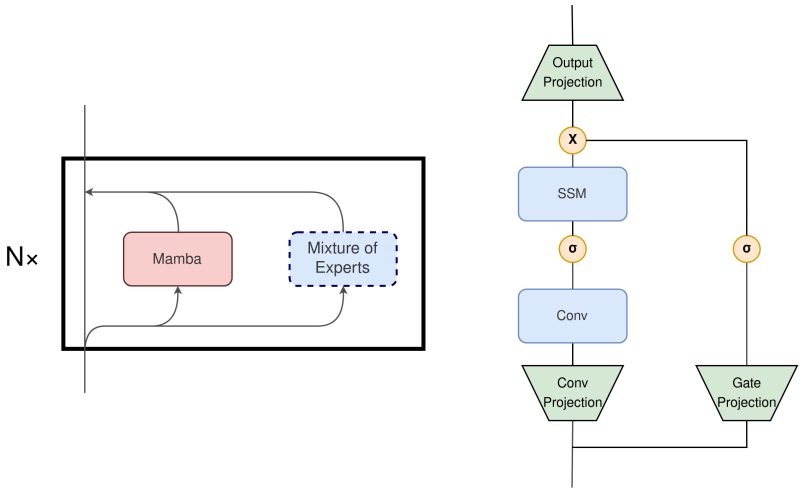


Figure 6.5: Diagram of Parallel MoE-Mamba architecture (left) and Mamba block (right). The outputs of the Gate and Conv Projections are E (expansion factor) times larger than the input, i.e., Conv and SSM operate on vectors $\in \mathbb{R}^{E \cdot d_{\text{model}}}$. Vanilla Mamba assumes $E = 2$ (Gu & Dao, 2023). The expansion factor E determines how much the input vector is scaled up by Gate and Conv Projection and then scaled down by Output Projection, and because of that, it is also proportional to the number of FLOPs and parameters in the Mamba layer.

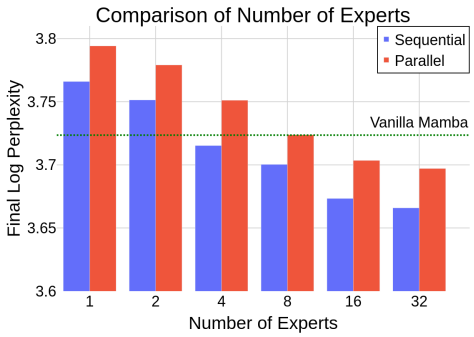


Figure 6.6: Final log perplexity comparison for varying numbers of experts in sequential and parallel MoE-Mamba.

# of Experts	MoE-Mamba	
	Sequential	Parallel
1	3.76	3.79
2	3.74	3.77
4	3.71	3.74
8	3.69	3.72
16	3.67	3.70
32	**3.66**	3.69

Table 6.6: Comparison of sequential and parallel MoE-Mamba—final log perplexity (1B tokens).

6.5.2 Modifying Mamba Block

Pursuing a uniform layer design, we experimented with replacing each of the three linear projections within the Mamba block with an MoE layer; see Figure 6.5. Enumerating all the possible placements results in $2^3 - 1 = 7$ possible designs (we discard one combination that would feature no MoE inside the block). We maintain a similar number of total parameters and FLOPs in all models by ensuring the total number of expert feed-forward layers in a block sums up to 24 regardless of the placement, i.e., the 24 experts are split evenly between one, two or three MoE's inside the block. Inspired by Fedus et al. (2022), we also performed experiments in which only half of the Mamba blocks were modified to include MoE, but the number of experts was increased to 48 to maintain the total number of parameters.

Three of the designs (Table 6.7) achieved marginally better results than vanilla Mamba, with none outperforming MoE-Mamba. These results suggest the most promising research directions in future work.

Model Name / Modified Projection	**MoE in Mamba**	
	All Layers	Every Other Layer
Vanilla Mamba	3.72	
MoE-Mamba (16 experts)	**3.67**	
Conv Projection	3.79	3.71
Gate Projection	3.89	3.70
Output Projection	4.05	3.70
Conv + Gate Projection	3.95	3.72
Conv + Output Projection	4.17	3.76
Gate + Output Projection	4.16	3.88
Conv + Gate + Output Projection	4.39	3.88

Table 6.7: Comparison of different variants of MoE in Mamba - final log perplexity (1B tokens).

6.6 Other Results

6.6.1 Relation between Speedup and Training Time

In our experiments, we notice that generally, as training continues, the speedup of MoE-Mamba compared to vanilla Mamba increases (see Figure 6.7). Specifically, the ratio

$$\text{speedup}(l) = \frac{\#\text{ processed tokens vanilla Mamba took to reach loss } l}{\#\text{ processed tokens MoE-Mamba took to reach loss } l}$$

increases as l decreases. The speedup in $\square_{25\text{M}}$ models oscillates between 1.6 and 1.9, while the speedup in $\square_{100\text{M}}$ models rises steadily.

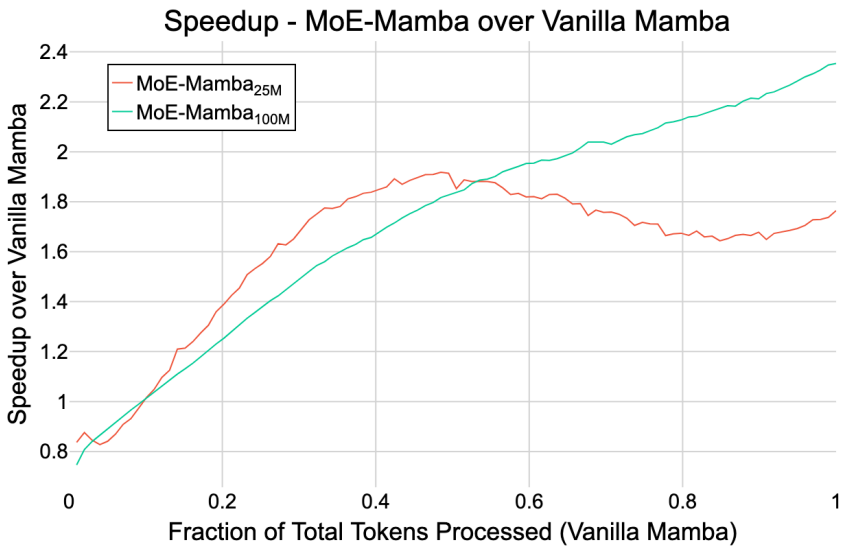


Figure 6.7: Speedup of different sizes of MoE-Mamba compared to their vanilla Mamba counterparts as training progresses.

6.6.2 Discrepancy between Accuracy and Perplexity.

We observed that throughout the training of a variant of one of our smaller models, MoE-Mamba$_{25M}$ with 32 instead of 42 experts as presented in Section 6.4.3, it maintains a lower perplexity than our strongest baseline (Transformer-MoE). However, at the same time, Transformer-MoE consistently achieves higher accuracy than MoE-Mamba. See Figure 6.8 for comparison.

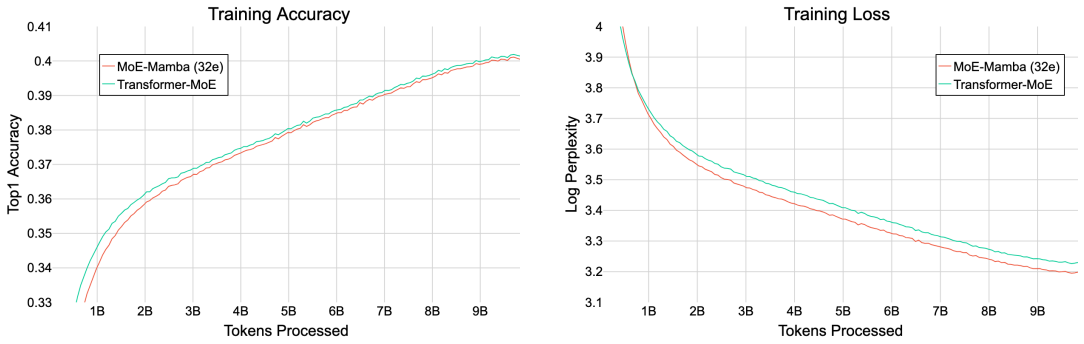


Figure 6.8: Discrepancy between accuracy and log perplexity: MoE-Mamba$_{25M}$ with 32 experts and Transformer$_{25M}$. Note that MoE-Mamba with 32 experts has fewer total parameters than the Transformer.

We hypothesize that this discrepancy hints at a potential failure mode of Mamba and other SSMs. Due to the compression of the history into a finite hidden state, their ability for verbatim token-copying is limited. On the other hand, the ability of the Transformer to copy verbatim, or more formally, predict the token $[B]$ given a prefix $...[A][B]...[A]$ (where $[A]$, $[B]$ can be any tokens) has been mechanistically studied by Elhage et al. (2021) and has been conjectured to be responsible for the Transformer's remarkable in-context learning capabilities (Olsson et al., 2022). This hypothesis is further supported by the gap in accuracy between models happening very early in training and diminishing later.

Intuitively, copying tokens can be learned more quickly using the attention mechanism, but this relative advantage diminishes with a better understanding of the language.

Similarly, Peng et al. (2023a) mentions that their attention-free model, RWKV, may have limited performance on tasks requiring the recall of precise information over long contexts due to a fixed-sized hidden state, a property that Mamba and other SSMs share. However, since the perplexity of Mamba can match the perplexity of a similarly sized Transformer, we suspect that Mamba compensates for that failure mode in other ways and might show a relative advantage on other tasks when compared to the Transformer. In particular, it might outperform Transformers in zero-shot tasks, in contrast to tasks allowing few-shot demonstrations or requiring in-context learning. We believe that performance in such tasks, dissimilar to copying, is more important for the future of language models.

6.6.3 Train and Test Set Performance

In the previous sections, we report the loss values obtained on the train set. Our training procedure samples from the dataset, so even without processing more tokens than in the C4 dataset, the same documents may be encountered multiple times. However, as we process a fraction of the tokens in the dataset, below 20% for our longest experiments, the difference in performance on the train set and the test set is negligible. For transparency, we provide the results on the test set as well in Figure 6.9. Their variance may be high due to a limited number of sequences in each evaluation step. Still, in all our experiments, their relative performance was the same in both the training and evaluation sets, but with more stable metrics on training. Therefore, we have decided to report a much more stable train set performance for more accurate comparisons.

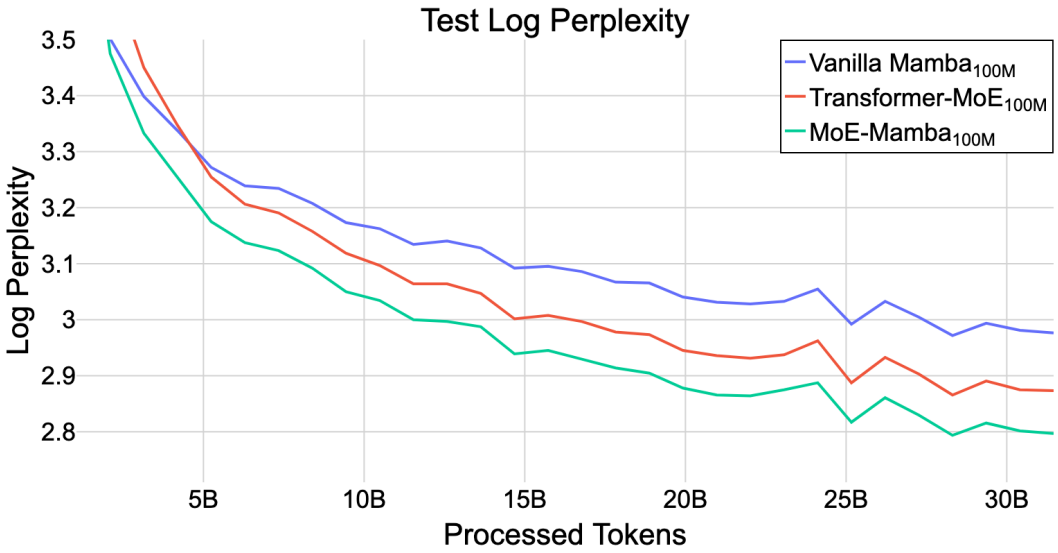


Figure 6.9: Test set loss.

6.7 Future Work and Limitations

Scaling. In this chapter, we conduct experiments on models with fewer than 1B active parameters per token, with total parameters up to 2.4B. Since MoE has enabled Transformers to scale to unprecedented sizes (Fedus et al., 2022), we are eager to observe the impact of scaling on MoE-Mamba approaches as well. Developing scaling laws will

be instrumental in this endeavor. After our work, Jamba (Lieber et al., 2024)—with a heterogeneous architecture interleaving Mixture of Experts, feed-forward, Mamba, and attention layers—showed continuous improvement when scaling to 52B total parameters. We believe that such hybrid architectures may provide the most efficient scaling capability, as they combine the strengths of each technique they incorporate. However, it remains to be seen if the additional complexity of such designs pays off.

Integrating MoE into the Mamba Layer. Our experiments show that interleaving the Mamba layer with a performant sparse MoE feed-forward layer results in a promising model. However, in the dense setting, Mamba performs slightly better without the feed-forward layer. This suggests that integrating sparse computation within the Mamba layer itself could yield even better results while conserving a simple, homogeneous architecture. Our experiments, detailed in Section 6.5.2, warrant some optimism, and we expect this line of research to remain relevant.

Exploration of Different Types of MoE in MoE-Mamba. While we base our design on the commonly used Switch (Fedus et al., 2022), numerous other MoE architectures have been proposed. Not only may those designs perform better overall, but it is possible that a different type of MoE will be optimal when combined with SSMs. Among possible changes in this regard are Expert-Choice routers (Zhou et al., 2022), fully differentiable architectures (Puigcerver et al., 2023) and Chapter 4, varying the granularity of experts (Chapter 3), and other modifications.

Distillation. Some works, like Fedus et al. (2022), have shown that MoE layers can be distilled back to feed-forward layers. We expect similar results for MoE-Mamba. Interestingly, the findings by Gu & Dao (2023) indicate that a Mamba module can emulate feed-forward layers effectively. This raises the question of whether MoE can be distilled into a vanilla Mamba module and how that can be achieved.

Synergies. We leave for future work more in-depth studies of synergies of Mamba and MoE. We suspect that there might be efficiency gains growing with the context length due to better hardware utilization; as for inference, Mamba alleviates computation and memory throughput issues stemming from larger context sizes, while MoE alleviates those same issues stemming from increasing number of parameters and knowledge stored in the model. This synergy may allow for unprecedented scaling of language models both in the number of parameters and length of the input/output.

Mamba and Attention Mechanism. Mamba and Transformers make different trade-offs during data processing, resulting in different sets of strengths and weaknesses. For example, Mamba can process very long inputs but might struggle with tasks requiring detailed knowledge of past input (e.g., some instances of copying). It would be interesting to explore combining these two architectures to achieve the best of both worlds.

Long Context Utilization. Mamba and other SSMs are praised for their ability to process long context. However, the extent to which they can utilize it effectively, and techniques for improving the utilization, have not yet been studied in depth. To that end, some methods developed for Transformers (Shi et al., 2023b; Tworkowski et al., 2023), including SPLiCe in Chapter 5, might be applicable.

Other Modalities. This work explores one direction in which Mamba can be extended. Mamba is a general architecture and is not limited to language modeling. We expect that it will be possible to apply MoE-Mamba to other tasks like non-textual sequence modeling presented by Gu & Dao (2023) and different modalities, such as vision, with initial work presented by Zhu et al. (2024).

6.8 Conclusions

This chapter introduced the first integration of Mixture of Experts with the Mamba architecture, termed MoE-Mamba. This novel method shares the inference benefits of Mamba while requiring 2.35× fewer training steps to reach the same performance. We demonstrated possible ways of combining these techniques and positively verified the performance improvements achieved through their combination. We confirmed with experiments on models up to 2.4B parameters and training lengths up to 30B tokens that these improvements over Mamba are robust to changes in model size, training duration, and number of experts.

In addition, we explored and evaluated numerous alternative designs integrating Mixture of Experts within the Mamba block. Although none of these variants outperformed MoE-Mamba, we believe these investigations can help prune ineffective research directions and highlight promising ones. Our work opens a new research direction of combining Mixture of Experts with State Space Models.

We believe that any future Large Language Model, regardless of the exact architecture, must employ Conditional Computation techniques to be truly efficient.

Chapter 7

Conclusions

In this work, we have shown multiple techniques for enhancing the efficiency of Large Language Models using Conditional Computation. In Chapter 2, we have demonstrated the potential of Conditional Computation during inference by introducing a Sparse Feed-Forward layer, with performance further supported by a novel Sparse QKV layer and augmented with other techniques to form a Terraformer. In Chapter 3, we introduced an improvement to Mixture of Experts—granularity. We also developed scaling laws, which both took granularity into account and corrected assumptions present in previous scaling laws for MoE. In Chapter 4, we have developed a continuous variant of the MoE layer, improving model stability while retaining the benefits of the sparse MoE. In Chapter 5, we have shown improvements in the long-context abilities of language models with relatively short fine-tuning, using structured packing of training examples. In Chapter 6, by integrating MoE into the recently developed Mamba architecture and achieving strong performance, we have demonstrated that the improvements of Conditional Computation seem independent of the Transformer architecture.

7.1 The Bitter Lesson

> "The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin."
> — Rich Sutton, The Bitter Lesson (Sutton, 2019)

We would like, in this chapter, to discuss The Bitter Lesson, as presented by Sutton (2019). It is clear that the success of Deep Learning, particularly the success of Large Language Models, can be attributed to the vast amount of computation they can leverage. In a world with underutilized compute, simpler architectures will win. Transformer's greatest strength, arguably, was its efficiency on hardware accelerators, as it eliminated the sequential processing of recurrent networks, which are inefficient on GPUs. GPT-3, when scaled to 175B parameters (3000× larger than the original Transformer's 65M), had a simpler architecture, using only the decoder and completely removing the encoder. This was also a strength, as the next-token prediction has high information density, with each word serving as both a datapoint and a label. Language modeling could also utilize an extremely large amount of data.

Scaling laws, as introduced by Kaplan et al. (2020), can be interpreted as a quantification of The Bitter Lesson. With them, we not only know which methods scale with available compute — we know which method scales better and by exactly how much. This was the reasoning behind our research work shown in Chapter 3 — through extrapolation, we aimed to show what is optimal to use at a scale unavailable with our academic budget.

However, scaling laws by themselves do not capture the full picture. In many cases, the saved FLOPs do not translate to the saved resources. And for this many different factors play a role - the efficiency of a given operation on a given hardware accelerator, the throughput or latency of memory access, the throughput between devices, possible optimizations to be made. In general, more complex architectures can offer better scaling in terms of FLOPs, but worse in terms of GPU-hours.

Future research endeavors on Large Language Models, if they want to have an impact, have to take into account both scaling capability and efficiency of their methods on real-world hardware. In particular, we would advocate for more research to be presented in the form of the scaling laws, with methods being compared in terms of their scalability. We also would advocate for better research on scaling laws themselves, especially on deriving them more reliably, more easily, using as little resources as possible. Just as well, work is needed on connecting scaling laws to metrics better than FLOPs in terms of performance on real hardware.

7.2 The Future

"Prediction is very difficult, especially if it is about the future."
— Niels Bohr

7.2.1 The Future of Conditional Computation in Language Modeling

We believe Conditional Computation is here to stay due to the efficiency it brings to language models. While there are cases where the additional memory footprint of MoE and similar techniques makes them infeasible, the overarching theme in recent years of Large Language Model development has been increased scale, bottlenecked by computational requirements rather than memory ones. We expect the best LLMs of the future to always involve some kind of Conditional Computation. Even in cases where the memory footprint prohibits Conditional Computation, we expect small dense models to generally be distilled from large MoE models instead of being trained from scratch.

Much research remains to be done in the field of Conditional Computation. For one, improving the hardware utilization of MoE models while retaining the expressiveness of techniques like fine-grained experts would be extremely useful for large-scale models. Such techniques need to consider not only performance on a single device or a single node of hardware accelerators but also communication latencies and throughput between machines. There is also a need to prevent load imbalance between experts on out-of-distribution data, or at least to alleviate its negative impacts during inference. Furthermore, better integration of Conditional Computation not only into replacements of feed-forward layers but also into token mixing layers (like attention or Mamba) would be extremely useful for scaling to longer contexts, as existing techniques have not found much use in industrial large-scale models.

7.2.2 The Future of Large Language Models

It is clear that Large Language Models will continue to grow in size and cost. There are too many research avenues for improving LLMs to enumerate here. Apart from important work on Conditional Computation and scaling laws mentioned in previous sections, we would like to highlight one in particular — data generation.

With constant scaling, the models will likely run out of data, as the amount of text on the Internet is finite. While there are vastly more bytes of data in different modalities, particularly video, we think that a tiny fraction (compared to text) of that data is information useful to learn. Even if it is not, the amount of video available for training is also limited. Therefore, we expect data generation to be important in the future, be it in a loop of generation and self-critique by the same model or data generated by interacting with the environment (probably virtual environment, e.g., programming). While in the past research showed the potential for model collapse with training on the same model's output (Shumailov et al., 2024), later works demonstrated the model collapse can be avoided (Gerstgrasser et al., 2024).

7.2.3 The Future of Artificial Intelligence

As the cost of model training and inference rises, we believe that LLMs will fortunately also grow in the value provided to the world. Based on our experience, the use of LLMs significantly accelerates the research process by enabling much faster implementation of experimental code, rapid data analysis and visualizations, as well as grammar and stylistic corrections while writing research papers.

On the other hand, there is growing concern among the research community and the public about the rising capabilities of AI models. These concerns most often cover current or near-term dangers with the use of AI, such as the spread of misinformation, increased surveillance, military use of AI, unemployment caused by automation, the reinforcement of biases present in training data, the potential for a single point of failure in autonomous systems, and the complex matters of privacy and copyright, among many other issues.

Some concerns involve long-term dangers. The primary issue is the development of Artificial General Intelligence (AGI) and the potential misalignment of AGI goals with human values, particularly in autonomous systems. These risks could lead to monetary losses, human casualties, and even the extinction of humanity. While the vision of an apocalypse may seem like science fiction, many AI researchers, including leaders of the largest AI labs, have signed the statement: "Mitigating the risk of extinction from AI should be a global priority alongside other societal-scale risks such as pandemics and nuclear war" (Center for AI Safety, 2023).

There are reasons for both caution and optimism. The potential catastrophic risks of AI development are being acknowledged and addressed by major AI labs (OpenAI, 2024; Anthropic, 2024; DeepMind, 2024). The potential upside of AGI is immense, as the automation of all possible types of work may bring unprecedented prosperity to humanity. We particularly look forward to autonomous scientific research, as it could accelerate the further improvement of AI itself and bring new inventions to all areas of life, including environmental and healthcare advancements.

Overall, we recognize the immense value of work on the safety and alignment of Artificial Intelligence to ensure it can benefit all, as well as the work on further development of AI efficiency and capabilities. We recommend cautious optimism about the future of AI.

Editorial Note

This thesis was edited and corrected for grammatical and stylistic errors using LLM-based systems. These systems also helped ensure the consistency and clarity of the text. We verified the suggested changes before incorporating them to ensure the meaning remained unchanged.

Acknowledgments

I want to express my sincere gratitude and appreciation to my PhD advisor, Marek Cygan, for his help in scaling the research team, providing fine-grained feedback, continuous support, long-context discussions, and guidance through the complex state spaces of research. Completing my PhD efficiently was conditioned on his mentorship.

I want to acknowledge and extend my appreciation to Henryk Michalewski, my advisor for the first two years of my PhD program, who directed me onto a fruitful research path before we parted ways.

I want to express my gratitude to all the members of LLM-Random, past and present. Building and working with this team has been one of the most rewarding experiences of my career. Thanks to: Michał Krutul, Maciej Pióro, Jan Małaśnicki, Jan Ludziejewski, Tomasz Odrzygóźdź, Szymon Antoniak, Krystian Król, Kamil Ciebiera, Maciej Stefaniak, Jakub Krajewski, Mikołaj Dziok, and Kamil Adamczewski.

I am grateful to many mentors, coworkers, and friends who helped me throughout my research career, motivated me, brainstormed ideas, shared intuitions, and supported me in numerous ways: Konrad Staniszewski, Tomasz Korbak, Sajid Siddiqi, Łukasz Kaiser, Piotr Miłoś, Aravindh Mahendran, Aakanksha Chowdhery, Piotr Padlewski, Dirk Weissenborn, Dominik Machoń, Piotr Migdał, Tomasz Trzciński, Iwona Pytel, Piotr Sankowski, Mehdi S. M. Sajjadi, Szymon Tworkowski, and Michał Łuszczyk.

I want to wholeheartedly thank Zuzanna Matuszewska and Paulina Kaczyńska for their tremendous support throughout my PhD. It would not have been possible without you.

Lastly, I want to thank my parents, Dorota and Dariusz Jaszczur, for everything they taught me and for the values they instilled in me.

The listing order is randomized within each paragraph.

This work was funded and supported by IDEAS NCBR, which also provided an excellent research environment, and the Polish National Science Center grant Preludium 2022/45/N/ST6/02222.

Appendix A: Sparse is Enough

A.1 Finetuning Terraformer on Summarization Task

We present a few examples of the abstracts generated by the Terraformer model for scientific papers in the arXiv dataset (Cohan et al., 2018). Table A.1 compares these abstracts to the corresponding examples from Tables I.25-27 in Section I of Zhang et al. (2020).

The abstracts are decoded using a greedy algorithm with temperature $T = 0.5$.

	ArXiv
Document (ID #34)	consider a set of objects which should be ranked on the basis of information about their bilateral relationships . similar problems arise , among others , in social choice theory xcite , sports xcite , psychology xcite , internet search xcite , and bibliometrics xcite . we discuss a universal version of the problem involving arbitrary preference intensities as well as incomplete and multiple comparisons . the main contribution of this paper is the presentation of an impossibility theorem : consistency requiring that if an object is ranked at least as high as another in two independent problems , then it is ranked as high as the other in the unified problem , too and self - consistency a less known but intuitive property , introduced in xcite , which prohibits to assign a lower rank for an object with a better or equivalent performance than another can not be met simultaneously by any ranking method on the set of all problems . domain restrictions and weakening of the properties are also investigated in order to get some positive results . since self - consistency is an axiom difficult to debate , we can not demand consistency from a ranking method . at the first glance , it is a somewhat surprising and controversial result making the aggregation of (incomplete) preferences impossible . one may argue that the whole concept of paired comparisons - based ranking is flawed . however , loss of consistency may be regarded as a necessary sacrifice for dimension reduction since the outcome should be an inherently transitive ranking even from intransitive data . this interpretation is reinforced by the connection between consistency and independence of irrelevant matches (which requires the relative ranking of two objects to be independent of the outcome of ... ...

Golden	assume a set of objects is given with information about their bilateral relationships , allowing for incomplete and multiple comparisons as well as different preference intensities . an axiomatic approach is applied for the problem of ranking the objects . consistency requires the preservation of relative ranking if two sets of such preferences are aggregated . self - consistency assigns the same rank for objects with the same performance , furthermore , an object should be ranked strictly higher if it shows an obviously better performance than another . it is revealed that these two properties can not be satisfied simultaneously . the impossibility holds under various restrictions on the domain . however , a positive result emerges if only the aggregation of ranking problems with the same comparison structure is allowed , for example , we have two round - robin tournaments .
Terraformer	we discuss a universal version of the problem involving arbitrary preference intensities as well as incomplete and multiple comparisons . the main contribution of this paper is the presentation of an impossibility theorem : consistency requiring that if an object is ranked at least as high as another in two independent problems , then it is ranked as high as the other in two independent problems , too and self - consistency a less known but intuitive property , introduced in @xcite . first , independence of irrelevant matches (posing a kind of independence concerning the relative ranking of two objects) is implied by three independent axioms , neutrality (independence of relabelling the objects) , symmetry (implying a flat ranking if all comparisons are drawn) , and consistency (perhaps the most natural property of additivity) . it turns out that consistency and self - consistency are still enough to get this result . the size of the problem does not solve the issue as it emerges with at least four objects . the size of the problem does not solve the issue as it emerges with at least four objects . the size of the problem is not a trivial extension of the aggregation of (incomplete) preferences .
ROUGE-1	41.87

Document (ID #289)	machine learning methods are used widely within high energy physics (hep) . one promising approach , used extensively outside of hep for applications such as handwriting recognition , is that of support vector machines (svms) , a supervised learning model used with associated learning algorithms for multivariate analysis (mva) . developed originally in the 1960s , with the current standard version proposed in 1995 xcite , svms aim to classify data points using a maximal margin hyperplane mapped from a linear classification problem to a possibly infinite dimensional hyperspace . however this means svms , like other mva classifiers , have a number of free parameters which need to be tuned on a case by case basis . this motivates a number methods for ensuring the classifier is sufficiently generalised such that when used on an unseen dataset the performance can be accurately predicted . in this paper a brief overview of svms is given in section [sec : svm] , with an example using svms shown in section [sec : checker] . generalisation is discussed in section [sec : gen] with an illustrative example of how this can improve performance given in section [sec : check2] . consider the problem of linear classification with the svm where the training set , xmath0 , is linearly separable . we define a separating hyperplane given by xmath1 , where xmath2 , the weight vector , is perpendicular to the hyperplane , and xmath3 , the bias , determines the distance of the hyperplane from the origin (fig .) . a decision function defined by xmath4 is used to positively and negatively classify xmath5 , the points in the training set . without further constraint the ... ...
Golden	we review the concept of support vector machines (svms) and discuss examples of their use . one of the benefits of svm algorithms , compared with neural networks and decision trees is that they can be less susceptible to over fitting than those other algorithms are to over training . this issue is related to the generalisation of a multivariate algorithm (mva) ; a problem that has often been overlooked in particle physics . we discuss cross validation and how this can be used to improve the generalisation of a mva in the context of high energy physics analyses . the examples presented use the toolkit for multivariate analysis (tmva) based on root and describe our improvements to the svm functionality and new tools introduced for cross validation within this framework .
Terraformer	support vector machines (svms) are used widely within high energy physics (hep) for applications such as handwriting recognition . the svm algorithm is a supervised learning model used with associated learning algorithms for multivariate analysis (mva) . the hard margin svm is an important svm algorithm , and the hard margin svm is an important svm algorithm . in this paper a brief overview of svms is presented , with an example showing similar performance to that of a bdt . however it is not clear without further checks as to whether the mvas are sufficiently generalised . hence a multistage cross - validation procedure has been outlined , which for the same example shows better performance as well as better agreement between the training and testing samples in the output distributions .

ROUGE-1	43.54
Document (ID #294)	the pure quantum states are traditionally associated with the wave function xcite or a vector in the hilbert space xcite . the mixed quantum states are described by the density matrix xcite or the density operator xcite . there exist several representations of quantum states in terms of the quasidistribution functions like the wigner function xcite and the husimi kano function xcite . the diagonal representation of quantum states was suggested in xcite (see also xcite) . it was studied and applied in xcite . in this representation , a quantum state is represented in terms of weighted sum of coherent - state xmath0 projectors . the properties of all the quantum - state representations considered are associated with the properties of the density operator which is hermitian , trace - class nonnegative operator . this means , in particular , that all the eigenvalues of the density operators must be nonnegative . in the quantum domain , the multipartite systems have a specific property connected with strong correlations of the quantum subsystems . this property provides the entanglement phenomenon xcite . in the diagonal representation of the density states , the weight function xmath1 is an analog of the probability - distribution function in the phase space . for some class of states , this function is identical to the probability - distribution function like in classical statistical mechanics . in xcite , the tomographic - probability representation of quantum states , where the quantum state is associated with the so - called symplectic tomogram , was introduced . the tomogram is a fair probability distribution containing the same information on quantum state that the density operator does (or such its characteristics as the wigner or husimi kano functions) . the aim of this work ... ...
Golden	the quasidistributions corresponding to the diagonal representation of quantum states are discussed within the framework of operator - symbol construction . the tomographic - probability distribution describing the quantum state in the probability representation of quantum mechanics is reviewed . the connection of the diagonal and probability representations is discussed . the superposition rule is considered in terms of the density - operator symbols . the separability and entanglement properties of multipartite quantum systems are formulated as the properties of the density - operator symbols of the system states .
Terraformer	the diagonal and probability representations of quantum states are reviewed . the superposition rule is formulated in both the diagonal and probability representations . the problem of separability and entanglement is formulated in both the diagonal and probability representations . the problem of separability and entanglement is formulated in both the diagonal and probability representations . the problem of separability and entanglement is formulated in both the diagonal and probability representations . the separability and entanglement properties of composed system in the tomographic and diagonal representations are analyzed .
ROUGE-2	27.16

Document (ID #32)	deep neural networks (dnns) have been receiving ubiquitous success in wide applications , ranging from computer vision xcite , to speech recognition xcite , natural language processing xcite , and domain adaptation xcite . as the sizes of data mount up , people usually have to increase the number of parameters in dnns so as to absorb the vast volume of supervision . high performance computing techniques are investigated to speed up dnn training , concerning optimization algorithms , parallel synchronisations on clusters w / o gpus , and stochastic binarization / ternarization , etc xcite . on the other hand the memory and energy consumption is usually , if not always , constrained in industrial applications xcite . for instance , for commercial search engines (e.g. , google and baidu) and recommendation systems (e.g. , netflix and youtube) , the ratio between the increased model size and the improved performance should be considered given limited online resources . compressing the model size becomes more important for applications on mobile and embedded devices xcite . having dnns running on mobile apps owns many great features such as better privacy , less network bandwidth and real time processing . however , the energy consumption of battery - constrained mobile devices is usually dominated by memory access , which would be greatly saved if a dnn model can fit in on - chip storage rather than dram storage (c.f . xcite for details) . a recent trend of studies are thus motivated to focus on compressing the size of dnns while mostly keeping their predictive performance xcite . with different intuitions , there are mainly two types of dnn compression methods , which could be used in conjunction for better parameter savings . the ... ...
Golden	as the complexity of deep neural networks (dnns) trend to grow to absorb the increasing sizes of data , memory and energy consumption has been receiving more and more attentions for industrial applications , especially on mobile devices . this paper presents a novel structure based on functional hashing to compress dnns , namely funhashnn . for each entry in a deep net , funhashnn uses multiple low - cost hash functions to fetch values in the compression space , and then employs a small reconstruction network to recover that entry . the reconstruction network is plugged into the whole network and trained jointly . funhashnn includes the recently proposed hashednets xcite as a degenerated case , and benefits from larger value capacity and less reconstruction loss . we further discuss extensions with dual space hashing and multi - hops . on several benchmark datasets , funhashnn demonstrates high compression ratios with little loss on prediction accuracy .

Terraformer	this paper presents a novel approach to relieve the hash collision effect through a sparse feature space . specifically , we use multiple hash functions to map per virtual entry into multiple values in compression space . then an additional network plays in a _ mapping function _ role from these hashednets to the virtual entry before hashing , which can be also regarded as ŏ047 reconstructing " the virtual entry from its multiple hashed values . plugged into and jointly trained within the original network , the reconstruction network is of a comparably ignorable size , i.e. , at low memory cost . this functional hashing structure includes hashednets as a degenerated special case , and facilitates less value collisions and better value reconstruction . experiments on several datasets demonstrate promisingly larger reduction of model sizes and/or less loss on prediction accuracy , compared with hashednets .
ROUGE-2	16.11
Document (ID #248)	stripped supernovae (sne) and long - duration gamma - ray bursts (long grbs) are nature s most powerful explosions from massive stars . they energize and enrich the interstellar medium , and , like beacons , they are visible over large cosmological distances . however , the mass and metallicity range of their progenitors is not known , nor the detailed physics of the explosion (see reviews by xcite and xcite) . stripped - envelope sne (i.e , sne of types iib , ib , and ic , e.g. , xcite) are core - collapse events whose massive progenitors have been stripped of progressively larger amounts of their outermost h and he envelopes (fig . [fig1]) . in particular , broad - lined sne ic (sne ic - bl) are sne ic whose line widths approach 20,000xmath030,000 xmath1 around maximum light (see below) and whose optical spectra show no trace of h and he . for the last 15 years , the exciting connection between long grbs and sne ic - bl , the only type of sne observed accompanying long grbs (for reviews , see xcite) , and the existence of many more sne ic - bl without grbs raises the question of what distinguishes sn - grb progenitors from those of ordinary sne ic - bl without grbs . viewing angle effects are probably not the reason why those sne ic - bl did not show an accompanied grbs xcite and based the same radio upper - limits , only xmath2 1% of sne ib / c appear to be accompanied by grbs xcite . one promising line of attack is to investigate what sets apart sne ib ... ...

Golden	while the connection between long gamma - ray bursts (grbs) and type ib / c supernovae (sne ib / c) from stripped stars has been well - established , one key outstanding question is what conditions and factors lead to each kind of explosion in massive stripped stars . one promising line of attack is to investigate what sets apart sne ib / c * with * grbs from those * without * grbs . here , i briefly present two observational studies that probe the sn properties and the environmental metallicities of sne ib / c (specifically broad - lined sne ic) with and without grbs . i present an analysis of expansion velocities based on published spectra and on the homogeneous spectroscopic cfa data set of over 70 sne of types iib , ib , ic and ic - bl , which triples the world supply of well - observed stripped sne . moreover , i demonstrate that a meta - analysis of the three published sn ib / c metallicity data sets , when including only values at the sn positions to probe natal oxygen abundances , indicates at very high significance that indeed sne ic erupt
Terraformer	the discovery of stripped supernovae (sne) and long - duration gamma - ray bursts (grbs) , and the discovery of long - duration gamma - ray bursts (grbs) , has raised the question of what distinguishes sne ic - bl from those of ordinary sne ic - bl without grbs . here i present a meta - analysis of published metallicities at the explosion site of stripped supernovae with and without grbs , and a meta - analysis of published metallicities at the explosion site of sne ib / c with and without grbs . the results of our analysis are compared with those of sne ic - bl without grbs , and with those of sne ic - bl without grbs . the results of our study are compared with those of sne ic - bl without grbs , and with sne ic - bl with grbs .
ROUGE-LSent	28.83

Document (ID #270)	the alice experiment is dedicated to the study of the properties of qcd matter created in nucleus - nucleus collisions at lhc energies xcite . the inner tracking system in the alice apparatus is made of position sensitive detectors which have to handle several thousands tracks per unit of rapidity . the two innermost layers at 3.9 xmath10 and 7.6 xmath10 radii , respectively , constitute the silicon pixel detector (spd) . the spatial precision and hit efficiency of the spd are key parameters since they determine the alice capability of detecting particles with open heavy - flavour xcite . + the basic detector unit of the alice spd is the ladder , a two - dimensional silicon matrix of pxmath11n reverse biased diodes of dimensions 50 x 425 xmath12 , flip - chip bonded to five read - out chips . each diode is connected to a cell of the front - end read - out asic via a pb - sn solder bump of 25 xmath13 diameter . the detector contains nearly 10xmath14 active cells in total . the read - out is binary . to reduce the material budget , the sensor thickness is limited to 200 xmath13 and the read - out chip wafers are thinned down to 150 xmath13 . further details can be found in xcite . + early prototypes of the alice spd elements , in the form of single - chip assemblies , were tested in high energy proton / pion beams at the cern sps in 2002 and 2003 . these assemblies were made with sensors of 200 xmath13 and 300 xmath13 thicknesses , while the read - out chips (unthinned) were 725 xmath13 thick . those beam tests were primarily aimed at evaluating the performance of ... ...
Golden	the two innermost layers of the alice inner tracking system are instrumented with silicon pixel detectors . single chip assembly prototypes of the alice pixels have been tested in high energy particle beams at the cern sps . detection efficiency and spatial precision have been studied as a function of the threshold and the track incidence angle . the experimental method , data analysis and main results are presented . d. elia@xmath0 , g. anelli@xmath1 , f. antinori@xmath2 , a. badal@xmath3 , g.e . bruno@xmath4 , m. burns@xmath1 , i.a . cali@xmath5 , m. campbell@xmath1 , m. caselle@xmath4 , s. ceresa@xmath1 , p. chochula@xmath1 , m. cinausero@xmath6 , j. conrad@xmath1 , r. dima@xmath2 , d. fabris@xmath2 , r.a . fini@xmath4 , e. fioretto@xmath6 , s. kapusta@xmath1 , a. kluge@xmath1 , m. krivda@xmath7 , v. lenti@xmath4 , f. librizzi@xmath3 , m. lunardon@xmath2 , v. manzari@xmath4 , m. morel@xmath1 , s. moretto@xmath2 , p. nilsson@xmath1 , f. osmic@xmath1 , g.s . pappalardo@xmath3 , v. paticchio@xmath4 , a. pepato@xmath2 , g. prete@xmath6 , a. pulvirenti@xmath3 , p. riedler@xmath1 , f. riggi@xmath3 , l. sndor@xmath7 , r. santoro@xmath4 , f. scarlassara@xmath2 , g. segato@xmath2 , f. soramel@xmath8 , g. stefanini@xmath1 , c. torcato de matos@xmath1 , r. turrisi@xmath2 , l. vannucci@xmath6 , g. viesti@xmath2 , t. virgili@xmath9 spatial precision , efficieny , silicon pixel detector , alice , lhc 29.40.gx , 29.40.wk

Terraformer	the performance of prototype assemblies for the alice silicon pixel detector have been extensively studied by using beam test data collected in the past years at the cern sps . the cluster pattern distribution , the intrinsic spatial precision and the detection efficiency have been investigated as a function of both the detector threshold and the incident angle of the tracks . the results show a very high detection efficiency (above 99%) in a wide threshold range and a spatial precision of about 10 @xmath0 in the short pixel side coordinate for normal track incidence and 210 dac threshold . the detector performance with angled tracks has also been investigated .
ROUGE-LSent	23.10

Table A.1: Abstracts generated by the Terraformer model on a set of examples of scientific papers from ArXiv. Note that the corresponding abstracts and scores for the Pegasus model are in Tables I.25-27 in section I of Zhang et al. (2020).

Appendix B: SPLiCe

B.1 Short Context Evaluation of 270M Models

In Table B.1, we assess the short context performance of 270M models from Section 5.5 and compare them against their starting checkpoint.

[htbp]

Tune Dataset	Method	arXiv	Code				Code & arXiv
			Haskell	Python	CUDA	All	
	Starting Chkpt	10.80	13.40	6.00	5.00	7.17	7.40
Wikipedia	SPLiCe bm25	10.64	13.28	6.00	5.00	7.16	7.38
	SPLiCe Cont	10.60	13.24	5.99	4.99	7.14	7.36
	Baseline	10.59	13.20	5.99	5.00	7.12	7.34
Stackexchange	SPLiCe bm25	9.28	12.94	5.91	4.89	7.05	7.19
	SPLiCe Cont	9.26	12.88	5.89	4.88	7.04	7.18
	Baseline	9.24	13.00	5.91	4.90	7.07	7.21
C	SPLiCe bm25	10.76	12.10	5.55	3.78	6.24	6.52
	SPLiCe Cont	10.76	12.06	5.53	3.76	6.21	6.50
	Baseline	10.73	11.96	5.49	3.68	6.16	6.44
	SPLiCe Repo	10.78	12.10	5.56	3.79	6.24	6.52
C#	SPLiCe bm25	10.86	12.74	5.73	4.67	6.72	6.98
	SPLiCe Cont	10.89	12.72	5.72	4.65	6.70	6.96
	Baseline	10.83	12.73	5.70	4.65	6.69	6.95
	SPLiCe Repo	10.91	12.77	5.74	4.68	6.72	6.98
Python	SPLiCe bm25	10.75	12.33	4.12	4.44	6.38	6.65
	SPLiCe Cont	10.76	12.28	4.09	4.43	6.36	6.64
	Baseline	10.72	12.19	3.98	4.40	6.33	6.61
	SPLiCe Repo	10.78	12.33	4.12	4.45	6.38	6.66

Table B.1: 2K context perplexity evaluation of models from Section 5.5.

B.2 Detailed Accuracy Improvements

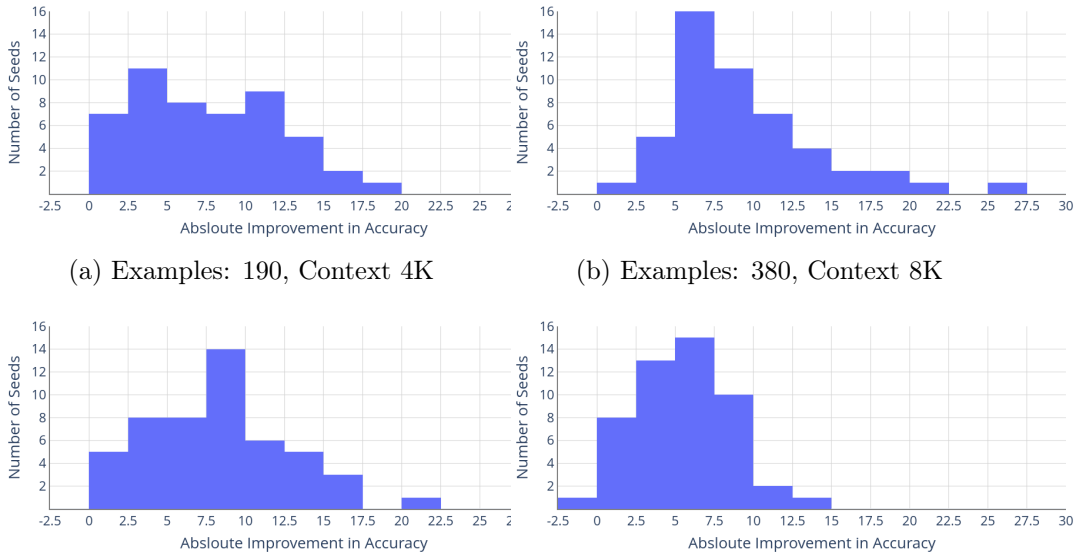
The performance of in-context learning heavily depends on the choice of in-context examples. To explore this in greater detail, we analyze the following random variable:

$$\Delta(c) = \mathrm{ACC}_{\textsc{SPLiCe}}(c) - \mathrm{ACC}_{\textsc{Baseline}}(c),$$

where $\mathrm{ACC}_{\textsc{SPLiCe}}(c)$ and $\mathrm{ACC}_{\textsc{Baseline}}(c)$ represent the accuracies of the model trained with SPLiCe and Baseline, respectively, on a random selection of in-context examples

c. Below, we present the histograms of $\delta(c)$. In Table 5.3 and Table 5.4, we report the mean Δ and 95% confidence intervals as Δ [confidence interval].

Figure B.1 provides additional details about accuracy improvements on TREC when considering different numbers of in-context examples.



(a) Examples: 190, Context 4K

(b) Examples: 380, Context 8K

(c) Examples: 780, Context 16K

(d) Examples: 1560, Context 32K

Figure B.1: Histograms of accuracy improvement of SPLICE BM25 over BASELINE on the TREC question classification task. The results are obtained by comparing the accuracy on the test set of TREC of the 3B FoT model trained with SPLICE to the model trained with the default data preparation method (BASELINE) across 50 sets of in-context examples. Each set of in-context examples consists of elements randomly sampled (without replacement) from the training subset of TREC. Note that the model trained with SPLICE is almost always better than the BASELINE.

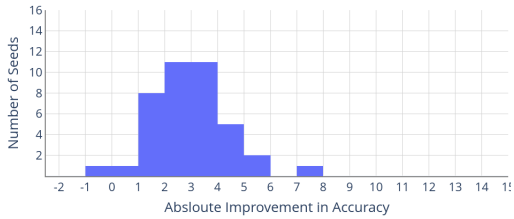
B.3 Detailed Results

In this section, we extend the results presented in Section 5.5.

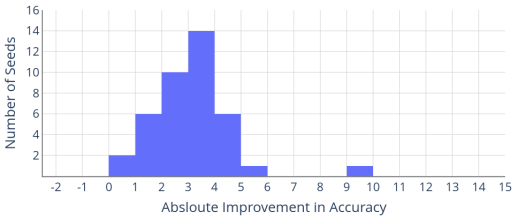
Tables B.2 and B.3 show the results of training a 270M parameter model for a 32K context on a 50/50 mixture of RedPajama data (organized in a standard way) and code data organized using a specified method. Table B.2 contains detailed results from training on C# and Python. Table B.3 contains results on C averaged across three different subsets of C. Both tables show that SPLICE outperforms the BASELINE by a significant margin.

The main advantage of SPLICE BM25/SPLICE CONT over the SPLICE REPO approach is its applicability to non-structured data. Table B.4 shows the detailed results of applying the SPLICE on non-code data. Note that training on non-code data allows us to improve the model's perplexity on the arXiv dataset compared to the model trained on code.

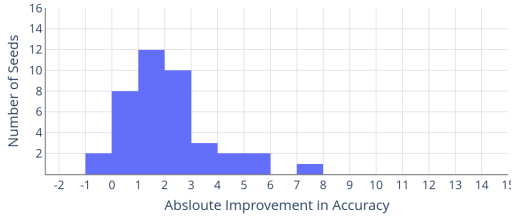
Tables B.5 and B.6 consider models trained with YaRN (Peng et al., 2023b), CodeLlama (Rozière et al., 2023), and Naive (no adjustment to RoPE) context extension methods. Table B.7 shows that a simple artificial extension of example length via random concatenation of documents (C source files) does not help.



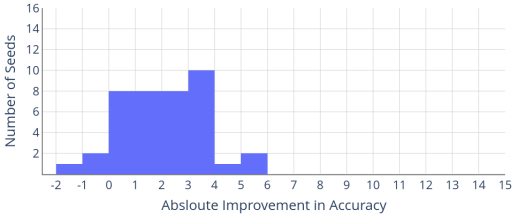
(a) Examples: 190, Context 16K, Model: 3B FoT



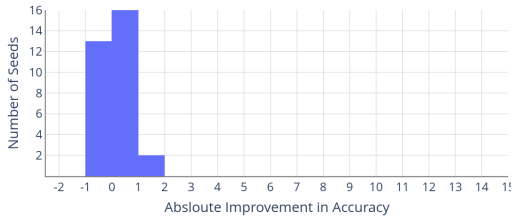
(b) Examples: 380, Context 32K, Model: 3B FoT



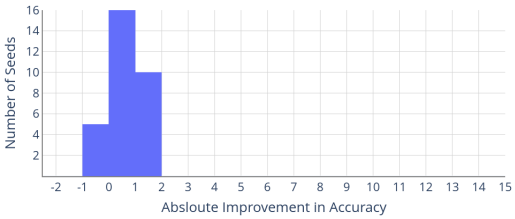
(c) Examples: 190, Context 16K, Model: 7B FoT



(d) Examples: 380, Context 32K, Model: 7B FoT



(e) Examples: 190, Context 16K, Model: 7B CL



(f) Examples: 380, Context 32K, Model: 7B CL

Figure B.2: Histograms of accuracy improvement of SPLiCe$_{\text{BM25}}$ over Baseline on DBPedia. We sample 40 sets of in-context examples and evaluate a random 500-element subset of the DBPedia test set for each set of in-context examples.

Altered Train Data: C#				
Eval/Method	SPLICE BM25	SPLICE CONT	BASELINE	SPLICE REPO
ArXiv	**5.52**	5.53	5.65	5.53
C	**2.39**	2.40	2.50	2.40
C++	**2.60**	2.61	2.74	2.62
CUDA	**2.46**	2.48	2.65	2.49
C#	**1.82**	**1.82**	1.90	**1.82**
Common Lisp	**3.41**	3.44	3.72	3.42
Dart	**2.14**	2.16	2.31	2.16
Emacs Lisp	**8.41**	8.46	8.85	**8.41**
Erlang	**2.80**	**2.80**	2.95	2.81
Fortran	**3.68**	3.71	4.05	3.72
Go	**1.94**	1.95	2.06	1.96
Groovy	**3.01**	3.03	3.25	3.04
Haskell	**3.33**	3.35	3.58	3.35
Java	**2.09**	2.10	2.22	2.10
Pascal	3.61	3.62	3.80	**3.60**
Python	**2.90**	2.91	3.07	2.91
Mean	**3.26**	3.27	3.46	3.27

Altered Train Data: Python				
Eval/Method	SPLICE BM25	SPLICE CONT	BASELINE	SPLICE REPO
ArXiv	**5.47**	5.49	5.57	5.48
C	**2.38**	2.39	2.45	2.39
C++	**2.61**	2.63	2.71	2.63
CUDA	**2.41**	2.43	2.56	2.44
C#	**1.98**	1.99	2.06	2.00
Common Lisp	**3.23**	3.28	3.47	3.27
Dart	**2.15**	2.17	2.27	2.17
Emacs Lisp	7.94	7.98	8.28	**7.93**
Erlang	**2.70**	2.71	2.82	2.71
Fortran	**3.42**	3.46	3.72	3.46
Go	**1.95**	1.96	2.03	1.96
Groovy	**2.97**	2.99	3.13	2.99
Haskell	**3.25**	3.28	3.46	3.27
Java	**2.12**	**2.12**	2.20	2.13
Pascal	**3.57**	3.58	3.73	3.58
Python	**2.53**	**2.53**	2.62	2.54
Mean	**3.17**	3.19	3.32	3.18

Table B.2: Perplexity results comparing different ways of organizing the same data. All runs started from the same 270M model with 2048 context and were trained for 32K context on a 50/50 mixture of RedPajama (organized in a standard way) and code organized in the mentioned ways. For details about training, please refer to Section 5.5.

	Altered Train Data: C			
Eval/Method	SPLICE BM25	SPLICE CONT	BASELINE	SPLICE REPO
ArXiv	**5.463** ± 0.002	5.477 ± 0.005	5.550 ± 0.002	5.474 ± 0.007
C	**2.126** ± 0.020	2.134 ± 0.020	2.173 ± 0.023	2.135 ± 0.020
C++	**2.396** ± 0.004	2.403 ± 0.002	2.467 ± 0.001	2.403 ± 0.006
CUDA	**2.219** ± 0.002	2.235 ± 0.005	2.330 ± 0.009	2.239 ± 0.004
C#	**1.939** ± 0.000	1.948 ± 0.001	2.016 ± 0.004	1.949 ± 0.002
Common Lisp	**2.994** ± 0.072	3.042 ± 0.091	3.195 ± 0.078	3.043 ± 0.102
Dart	**2.141** ± 0.002	2.155 ± 0.004	2.268 ± 0.009	2.156 ± 0.002
Emacs Lisp	7.857 ± 0.017	7.851 ± 0.020	8.098 ± 0.027	**7.840** ± 0.019
Erlang	**2.665** ± 0.002	2.680 ± 0.003	2.769 ± 0.007	2.676 ± 0.003
Fortran	**3.306** ± 0.010	3.335 ± 0.010	3.554 ± 0.010	3.333 ± 0.012
Go	**1.910** ± 0.001	1.924 ± 0.007	1.999 ± 0.002	1.924 ± 0.006
Groovy	**3.009** ± 0.001	3.026 ± 0.006	3.147 ± 0.013	3.025 ± 0.007
Haskell	**3.198** ± 0.001	3.221 ± 0.001	3.371 ± 0.008	3.220 ± 0.008
Java	**2.075** ± 0.002	2.086 ± 0.001	2.161 ± 0.006	2.085 ± 0.005
Pascal	**3.492** ± 0.026	3.496 ± 0.019	3.622 ± 0.021	3.513 ± 0.014
Python	**2.810** ± 0.002	2.824 ± 0.001	2.931 ± 0.008	2.827 ± 0.006
Mean	**3.100** ± 0.004	3.115 ± 0.006	3.228 ± 0.005	3.115 ± 0.009

Table B.3: To assess the statistical significance of our results, we prepare three subsets of C and train the models on a 50/50 mixture of RedPajama data (organized in the standard way) and C data organized using one of the methods. Note that the standard deviation is much lower than the perplexity improvements from using SPLICE.

Altered Train Data: StackExchange			
Eval/Method	SPLICE BM25	SPLICE CONT	BASELINE
ArXiv	**5.07**	5.09	5.14
C	**2.68**	2.69	2.70
C++	**3.02**	3.04	3.06
CUDA	**2.89**	2.93	2.94
C#	**2.27**	2.28	2.29
Common Lisp	**4.02**	4.06	4.08
Dart	**2.58**	2.60	2.61
Emacs Lisp	**9.55**	9.67	9.69
Erlang	**3.13**	3.16	3.18
Fortran	**4.28**	4.34	4.38
Go	**2.24**	2.25	2.27
Groovy	**3.62**	3.66	3.68
Haskell	**3.88**	3.91	3.94
Java	**2.43**	2.45	2.45
Pascal	**4.08**	4.11	4.14
Python	**3.32**	3.35	3.36
Mean	**3.69**	3.73	3.74

Altered Train Data: Wikipedia			
Eval/Method	SPLICE BM25	SPLICE CONT	BASELINE
ArXiv	**5.64**	5.65	5.73
C	**2.65**	2.67	2.71
C++	**2.98**	3.01	3.07
CUDA	**2.87**	2.92	3.00
C#	**2.22**	2.24	2.29
Common Lisp	**3.87**	3.96	4.08
Dart	**2.51**	2.55	2.61
Emacs Lisp	**9.38**	9.45	9.63
Erlang	**3.13**	3.16	3.23
Fortran	**4.23**	4.32	4.49
Go	**2.18**	2.21	2.26
Groovy	**3.49**	3.55	3.67
Haskell	**3.82**	3.87	3.97
Java	**2.39**	2.41	2.46
Pascal	4.32	**4.23**	4.40
Python	**3.26**	3.30	3.37
Mean	**3.68**	3.72	3.81

Table B.4: Perplexity results comparing different methods of organizing the same data. All runs started from the same $270M$ model with a 2048 context and were trained for a 32K context on a 50/50 mixture of RedPajama (organized in a standard manner) and other data organized using one of the methods. For details about training, please refer to Section 5.5. Note that the model trained with SPLICE on StackExchange outperforms the one trained on code on the arXiv evaluation, showing the benefits of SPLICE's applicability to non-code data.

Altered Train Data: C# Context 16K: CodeLlama			
Eval/Method	SPLICE BM25	BASELINE	SPLICE REPO
ArXiv	**5.74**	5.76	**5.74**
C	**2.66**	2.70	**2.66**
C++	**2.79**	2.83	**2.79**
CUDA	**2.53**	2.58	2.54
C#	**1.91**	1.93	**1.91**
Common Lisp	**3.78**	3.85	3.79
Dart	**2.28**	2.33	**2.28**
Emacs Lisp	**8.29**	8.41	8.30
Erlang	**3.57**	3.64	3.58
Fortran	**3.93**	4.01	3.95
Go	**1.99**	2.03	2.00
Groovy	**2.95**	3.01	2.96
Haskell	**4.28**	4.37	**4.28**
Java	**2.31**	2.35	**2.31**
Pascal	**3.67**	3.72	**3.67**
Python	**3.22**	3.27	**3.22**
Mean	**3.49**	3.55	3.50
Context 16K: YaRN			
Eval/Method	SPLICE BM25	BASELINE	SPLICE REPO
ArXiv	**5.77**	5.79	**5.77**
C	**2.68**	2.72	**2.68**
C++	**2.81**	2.85	**2.81**
CUDA	**2.55**	2.61	2.56
C#	**1.92**	1.94	**1.92**
Common Lisp	**3.83**	3.92	3.84
Dart	**2.30**	2.36	**2.30**
Emacs Lisp	**8.37**	8.52	8.38
Erlang	**3.60**	3.67	3.61
Fortran	**3.97**	4.06	3.99
Go	**2.00**	2.04	2.01
Groovy	**2.98**	3.03	**2.98**
Haskell	**4.32**	4.42	**4.32**
Java	**2.33**	2.37	**2.33**
Pascal	**3.69**	3.75	**3.69**
Python	**3.24**	3.29	**3.24**
Mean	**3.52**	3.58	3.53

Table B.5: Perplexity results comparing different ways of organizing the same data for non-FoT models. All runs started from the same 270M model with a 2048 context and were trained for a 16K context on a 50/50 mixture of RedPajama (organized in a standard way) and C# code organized in one of three ways.

Altered Train Data: C# Context 16K: Naive			
Eval/Method	SPLICE BM25	BASELINE	SPLICE REPO
ArXiv	**6.25**	6.33	**6.25**
C	**2.88**	2.96	2.89
C++	**3.04**	3.13	3.05
CUDA	**2.84**	2.96	2.85
C#	**2.04**	2.08	**2.04**
Common Lisp	4.40	4.56	**4.39**
Dart	**2.50**	2.60	2.51
Emacs Lisp	**9.25**	9.46	**9.25**
Erlang	**3.98**	4.10	4.00
Fortran	**4.56**	4.79	4.59
Go	**2.14**	2.21	2.16
Groovy	**3.27**	3.39	3.28
Haskell	**4.84**	5.03	4.87
Java	**2.52**	2.60	2.53
Pascal	**4.05**	4.20	4.10
Python	**3.55**	3.66	3.56
Mean	**3.88**	4.00	3.89

Table B.6: Perplexity results comparing different ways of organizing the same data for non-FoT models. All runs started from the same 270M model with 2048 context and were trained for 16K context on a 50/50 mixture of RedPajama with a Naive context extension method.

	Altered Train Data: C		
Eval/Method	SPLiCe BM25	Baseline	DomRnd
ArXiv	**5.46**	5.55	5.55
C	**2.13**	2.17	2.18
C++	**2.40**	2.47	2.47
CUDA	**2.22**	2.33	2.33
C#	**1.94**	2.02	2.02
Common Lisp	**2.99**	3.20	3.18
Dart	**2.14**	2.27	2.27
Emacs Lisp	**7.86**	8.10	8.09
Erlang	**2.67**	2.77	2.77
Fortran	**3.31**	3.55	3.56
Go	**1.91**	2.00	2.00
Groovy	**3.01**	3.15	3.15
Haskell	**3.20**	3.37	3.37
Java	**2.07**	2.16	2.16
Pascal	**3.49**	3.62	3.64
Python	**2.81**	2.93	2.93
Mean	**3.10**	3.23	3.23

Table B.7: Perplexity results comparing different methods of organizing the same data. All runs started from the same 270M model with a 2048 context and were trained for 32K context on a 50/50 mixture of RedPajama (organized in a standard way) and C code organized in one of three ways.

Bibliography

Ainslie, J., Lei, T., de Jong, M., Ontañón, S., Brahma, S., Zemlyanskiy, Y., Uthus, D. C., Guo, M., Lee-Thorp, J., Tay, Y., Sung, Y., and Sanghai, S. Colt5: Faster long-range transformers with conditional computation. *CoRR*, abs/2303.09752, 2023. doi: 10.48550/arXiv.2303.09752. URL https://doi.org/10.48550/arXiv.2303.09752.

Allal, L. B., Li, R., Kocetkov, D., Mou, C., Akiki, C., Ferrandis, C. M., Muennighoff, N., Mishra, M., Gu, A., Dey, M., Umapathi, L. K., Anderson, C. J., Zi, Y., Poirier, J. L., Schoelkopf, H., Troshin, S., Abulkhanov, D., Romero, M., Lappert, M., Toni, F. D., del Río, B. G., Liu, Q., Bose, S., Bhattacharyya, U., Zhuo, T. Y., Yu, I., Villegas, P., Zocca, M., Mangrulkar, S., Lansky, D., Nguyen, H., Contractor, D., Villa, L., Li, J., Bahdanau, D., Jernite, Y., Hughes, S., Fried, D., Guha, A., de Vries, H., and von Werra, L. Santacoder: don't reach for the stars!, 2023.

Anthony, Q., Tokpanov, Y., Glorioso, P., and Millidge, B. Blackmamba: Mixture of experts for state-space models, 2024.

Anthropic. Model card and evaluations for claude models. Technical report, Anthropic, 2023. URL https://www-files.anthropic.com/production/images/Model-Card-Claude-2.pdf.

Anthropic. Responsible scaling policy, 2024. URL https://www-cdn.anthropic.com/1adf000c8f675958c2ee23805d91aaade1cd4613/responsible-scaling-policy.pdf. Accessed: 2024-09-17.

Azerbayev, Z., Piotrowski, B., and Avigad, J. Proofnet: A benchmark for autoformalizing and formally proving undergraduate-level mathematics problems. In *Advances in Neural Information Processing Systems 35, 2nd MATH-AI Workshop at NeurIPS'22*, 2022. URL https://mathai2022.github.io/papers/20.pdf.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate, 2016.

Barrabi, T. Openai may reportedly lose $5b this year alone on massive chatgpt costs, 2024. URL https://nypost.com/2024/07/25/business/openai-may-lose-5b-this-year-alone-on-chatgpt-costs-report/. Accessed: 2024-09-17.

Bassani, E. retriv: A Python Search Engine for the Common Man, May 2023. URL https://github.com/AmenRa/retriv.

Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., van den Driessche, G., Lespiau, J., Damoc, B., Clark, A., de Las Casas, D., Guy, A., Menick,

J., Ring, R., Hennigan, T., Huang, S., Maggiore, L., Jones, C., Cassirer, A., Brock, A., Paganini, M., Irving, G., Vinyals, O., Osindero, S., Simonyan, K., Rae, J. W., Elsen, E., and Sifre, L. Improving language models by retrieving from trillions of tokens. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 2206–2240. PMLR, 2022. URL https://proceedings.mlr.press/v162/borgeaud22a.html.

Brent, R. P. An algorithm with guaranteed convergence for finding a zero of a function. *Comput. J.*, 14:422–425, 1971. URL https://api.semanticscholar.org/CorpusID:10312755.

Brix, C., Bahar, P., and Ney, H. Successfully applying the stabilized lottery ticket hypothesis to the transformer architecture. *arXiv preprint arXiv:2005.03454*, 2020.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners, 2020.

Center for AI Safety. Statement on ai risk. https://www.safe.ai/work/statement-on-ai-risk, 2023. Accessed: 2024-09-17.

Cerisara, C. *SlowLLM: large language models on consumer hardware*. PhD thesis, CNRS, 2023.

Chan, S., Santoro, A., Lampinen, A. K., Wang, J., Singh, A., Richemond, P. H., McClelland, J. L., and Hill, F. Data distributional properties drive emergent in-context learning in transformers. In *NeurIPS*, 2022.

Chen, S., Wong, S., Chen, L., and Tian, Y. Extending context window of large language models via positional interpolation, 2023.

Chi, Z., Dong, L., Huang, S., Dai, D., Ma, S., Patra, B., Singhal, S., Bajaj, P., Song, X., Mao, X.-L., Huang, H., and Wei, F. On the representation collapse of sparse mixture of experts, 2022.

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

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A. M., Pillai, T. S., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., and Fiedel, N. Palm: Scaling language modeling with pathways, 2022.

Clark, A., de las Casas, D., Guy, A., Mensch, A., Paganini, M., Hoffmann, J., Damoc, B., Hechtman, B., Cai, T., Borgeaud, S., van den Driessche, G., Rutherford, E., Hennigan, T., Johnson, M., Millican, K., Cassirer, A., Jones, C., Buchatskaya, E., Budden, D., Sifre, L., Osindero, S., Vinyals, O., Rae, J., Elsen, E., Kavukcuoglu, K., and Simonyan, K. Unified scaling laws for routed language models, 2022.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems, 2021.

Cohan, A., Dernoncourt, F., Kim, D. S., Bui, T., Kim, S., Chang, W., and Goharian, N. A discourse-aware attention model for abstractive summarization of long documents. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 2018. doi: 10.18653/v1/n18-2097. URL `http://dx.doi.org/10.18653/v1/n18-2097`.

Dai, D., Dong, L., Ma, S., Zheng, B., Sui, Z., Chang, B., and Wei, F. Stablemoe: Stable routing strategy for mixture of experts, 2022.

Dai, D., Deng, C., Zhao, C., Xu, R. X., Gao, H., Chen, D., Li, J., Zeng, W., Yu, X., Wu, Y., Xie, Z., Li, Y. K., Huang, P., Luo, F., Ruan, C., Sui, Z., and Liang, W. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models, 2024.

Dasigi, P., Lo, K., Beltagy, I., Cohan, A., Smith, N. A., and Gardner, M. A dataset of information-seeking questions and answers anchored in research papers, 2021.

de Vries, H. In the long (context) run, 2023. URL `https://www.harmdevries.com/post/context-length/`. Accessed: 2024-09-17.

DeepMind. Frontier safety framework, 2024. URL `https://storage.googleapis.com/deepmind-media/DeepMind.com/Blog/introducing-the-frontier-safety-framework/fsf-technical-report.pdf`. Accessed: 2024-09-17.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, Ł. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Du, N., Huang, Y., Dai, A. M., Tong, S., Lepikhin, D., Xu, Y., Krikun, M., Zhou, Y., Yu, A. W., Firat, O., Zoph, B., Fedus, L., Bosma, M., Zhou, Z., Wang, T., Wang, Y. E., Webster, K., Pellat, M., Robinson, K., Meier-Hellstern, K., Duke, T., Dixon, L., Zhang, K., Le, Q. V., Wu, Y., Chen, Z., and Cui, C. Glam: Efficient scaling of language models with mixture-of-experts, 2022.

Eigen, D., Ranzato, M., and Sutskever, I. Learning factored representations in a deep mixture of experts, 2014.

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., DasSarma, N., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. A mathematical framework

for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022.

Frantar, E., Riquelme, C., Houlsby, N., Alistarh, D., and Evci, U. Scaling laws for sparsely-connected foundation models, 2023.

François. *The Little Book of Deep Learning*. Writers Republic LLC, Jun 20, 2023, 2023. ISBN 9732346493, 9789732346495.

Fu, D. Y., Dao, T., Saab, K. K., Thomas, A. W., Rudra, A., and Ré, C. Hungry hungry hippos: Towards language modeling with state space models, 2023.

Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Gale, T., Narayanan, D., Young, C., and Zaharia, M. Megablocks: Efficient sparse training with mixture-of-experts. In Song, D., Carbin, M., and Chen, T. (eds.), *Proceedings of Machine Learning and Systems*, volume 5, pp. 288–304. Curan, 2023. URL `https://proceedings.mlsys.org/paper_files/paper/2023/file/5a54f79333768effe7e8927bcccffe40-Paper-mlsys2023.pdf`.

Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., Phang, J., Reynolds, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, sep 2021. URL `https://doi.org/10.5281/zenodo.5371628`.

Geng, X. Easylm: A simple and scalable training framework for large language models, March 2023. URL `https://github.com/young-geng/EasyLM`.

Geng, X. and Liu, H. Openllama: An open reproduction of llama, May 2023. URL `https://github.com/openlm-research/open_llama`.

Gerstgrasser, M., Schaeffer, R., Dey, A., Rafailov, R., Sleight, H., Hughes, J., Korbak, T., Agrawal, R., Pai, D., Gromov, A., Roberts, D. A., Yang, D., Donoho, D. L., and Koyejo, S. Is model collapse inevitable? breaking the curse of recursion by accumulating real and synthetic data, 2024. URL `https://arxiv.org/abs/2404.01413`.

Ghorbani, B., Firat, O., Freitag, M., Bapna, A., Krikun, M., Garcia, X., Chelba, C., and Cherry, C. Scaling laws for neural machine translation, 2021.

Gidiotis, A. and Tsoumakas, G. A divide-and-conquer approach to the summarization of long documents. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:3029–3040, 2020. doi: 10.1109/TASLP.2020.3037401.

Groeneveld, D., Beltagy, I., Walsh, P., Bhagia, A., Kinney, R., Tafjord, O., Jha, A. H., Ivison, H., Magnusson, I., Wang, Y., Arora, S., Atkinson, D., Authur, R., Chandu, K. R., Cohan, A., Dumas, J., Elazar, Y., Gu, Y., Hessel, J., Khot, T., Merrill, W., Morrison, J., Muennighoff, N., Naik, A., Nam, C., Peters, M. E., Pyatkin, V., Ravichander, A., Schwenk, D., Shah, S., Smith, W., Strubell, E., Subramani, N., Wortsman, M., Dasigi,

P., Lambert, N., Richardson, K., Zettlemoyer, L., Dodge, J., Lo, K., Soldaini, L., Smith, N. A., and Hajishirzi, H. Olmo: Accelerating the science of language models, 2024.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces, 2023.

Gu, A., Johnson, I., Goel, K., Saab, K., Dao, T., Rudra, A., and Ré, C. Combining recurrent, convolutional, and continuous-time models with linear state-space layers, 2021.

Gu, A., Goel, K., Gupta, A., and Ré, C. On the parameterization and initialization of diagonal state space models. *Advances in Neural Information Processing Systems*, 35:35971–35983, 2022a.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces, 2022b.

Gu, Y., Dong, L., Wei, F., and Huang, M. Pre-training to learn in context. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4849–4870, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.267. URL `https://aclanthology.org/2023.acl-long.267`.

Gupta, A., Gu, A., and Berant, J. Diagonal state spaces are as effective as structured state spaces. *Advances in Neural Information Processing Systems*, 35:22982–22994, 2022.

Gupta, M. and Agrawal, P. Compression of deep learning models for text: A survey. *arXiv preprint arXiv:2008.05221*, 2020.

Han, X., Simig, D., Mihaylov, T., Tsvetkov, Y., Celikyilmaz, A., and Wang, T. Understanding in-context learning via supportive pretraining data, 2023.

Hazimeh, H., Zhao, Z., Chowdhery, A., Sathiamoorthy, M., Chen, Y., Mazumder, R., Hong, L., and Chi, E. H. Dselect-k: Differentiable selection in the mixture of experts with applications to multi-task learning, 2021.

He, T., Tan, X., Xia, Y., He, D., Qin, T., Chen, Z., and Liu, T.-Y. Layer-wise coordination between encoder and decoder for neural machine translation. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31, 2018.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding, 2021.

Henighan, T., Kaplan, J., Katz, M., Chen, M., Hesse, C., Jackson, J., Jun, H., Brown, T. B., Dhariwal, P., Gray, S., Hallacy, C., Mann, B., Radford, A., Ramesh, A., Ryder, N., Ziegler, D. M., Schulman, J., Amodei, D., and McCandlish, S. Scaling laws for autoregressive generative modeling, 2020.

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., Rae, J. W., Vinyals, O., and Sifre, L. Training compute-optimal large language models, 2022.

Hovy, E., Gerber, L., Hermjakob, U., Lin, C.-Y., and Ravichandran, D. Toward semantics-based answer pinpointing. In *Proceedings of the First International Conference on Human Language Technology Research*, 2001. URL https://www.aclweb.org/anthology/H01-1069.

Huber, P. J. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 35(1):73 – 101, 1964. doi: 10.1214/aoms/1177703732. URL https://doi.org/10.1214/aoms/1177703732.

Iandola, F. N., Shaw, A. E., Krishna, R., and Keutzer, K. W. Squeezebert: What can computer vision teach nlp about efficient neural networks? *arXiv preprint arXiv:2006.11316*, 2020.

Izacard, G., Caron, M., Hosseini, L., Riedel, S., Bojanowski, P., Joulin, A., and Grave, E. Unsupervised dense information retrieval with contrastive learning. *Trans. Mach. Learn. Res.*, 2022, 2022. URL https://openreview.net/forum?id=jKN1pXi7b0.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991. doi: 10.1162/neco.1991.3.1.79.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. Mistral 7b, 2023.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., de las Casas, D., Hanna, E. B., Bressand, F., Lengyel, G., Bour, G., Lample, G., Lavaud, L. R., Saulnier, L., Lachaux, M.-A., Stock, P., Subramanian, S., Yang, S., Antoniak, S., Scao, T. L., Gervet, T., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. Mixtral of experts, 2024.

Johnson, J., Douze, M., and Jégou, H. Billion-scale similarity search with gpus, 2017.

Kaiser, L. and Bengio, S. Discrete autoencoders for sequence models. *arXiv preprint arXiv:1801.09797*, 2018.

Kaiser, Ł. and Sutskever, I. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models, 2020.

Karpathy, A. nanogpt, 2022. URL https://github.com/karpathy/nanoGPT.

Kitaev, N., Kaiser, Ł., and Levskaya, A. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. Albert: A lite bert for self-supervised learning of language representations, 2020.

Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., and Bizer, C. Dbpedia - A large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015. doi: 10.3233/SW-140134. URL https://doi.org/10.3233/SW-140134.

Lei, T., Zhang, Y., and Artzi, Y. Training rnns as fast as cnns. *CoRR*, abs/1709.02755, 2017. URL http://arxiv.org/abs/1709.02755.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. Gshard: Scaling giant models with conditional computation and automatic sharding, 2020.

Levine, Y., Wies, N., Jannai, D., Navon, D., Hoshen, Y., and Shashua, A. The inductive bias of in-context learning: Rethinking pretraining example design. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=lnEaqbTJIRz.

Lewis, M., Bhosale, S., Dettmers, T., Goyal, N., and Zettlemoyer, L. Base layers: Simplifying training of large, sparse models, 2021.

Li, B., Kong, Z., Zhang, T., Li, J., Li, Z., Liu, H., and Ding, C. Efficient transformer-based large scale language representations using hardware-friendly block structured pruning. *arXiv preprint arXiv:2009.08065*, 2020a.

Li, C. Estimating the cost to train gpt-3, 2020. URL https://lambdalabs.com/blog/demystifying-gpt-3. Accessed: 2024-09-17.

Li, D., Shao, R., Xie, A., Sheng, Y., Zheng, L., Gonzalez, J. E., Stoica, I., Ma, X., and Zhang, H. How long can open-source llms truly promise on context length?, June 2023a. URL https://lmsys.org/blog/2023-06-29-longchat.

Li, R., Allal, L. B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., Liu, Q., Zheltonozhskii, E., Zhuo, T. Y., Wang, T., Dehaene, O., Davaadorj, M., Lamy-Poirier, J., Monteiro, J., Shliazhko, O., Gontier, N., Meade, N., Zebaze, A., Yee, M., Umapathi, L. K., Zhu, J., Lipkin, B., Oblokulov, M., Wang, Z., V, R. M., Stillerman, J., Patel, S. S., Abulkhanov, D., Zocca, M., Dey, M., Zhang, Z., Moustafa-Fahmy, N., Bhattacharyya, U., Yu, W., Singh, S., Luccioni, S., Villegas, P., Kunakov, M., Zhdanov, F., Romero, M., Lee, T., Timor, N., Ding, J., Schlesinger, C., Schoelkopf, H., Ebert, J., Dao, T., Mishra, M., Gu, A., Robinson, J., Anderson, C. J., Dolan-Gavitt, B., Contractor, D., Reddy, S., Fried, D., Bahdanau, D., Jernite, Y., Ferrandis, C. M., Hughes, S., Wolf, T., Guha, A., von Werra, L., and de Vries, H. Starcoder: may the source be with you! *CoRR*, abs/2305.06161, 2023b. doi: 10.48550/arXiv.2305.06161. URL https://doi.org/10.48550/arXiv.2305.06161.

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X., and Yan, X. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *arXiv preprint arXiv:1907.00235*, 2019.

Li, X. and Roth, D. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002. URL https://www.aclweb.org/anthology/C02-1150.

Li, Y., Cai, T., Zhang, Y., Chen, D., and Dey, D. What makes convolutional models great on long sequence modeling? *arXiv preprint arXiv:2210.09298*, 2022.

Li, Z., Wallace, E., Shen, S., Lin, K., Keutzer, K., Klein, D., and Gonzalez, J. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In *International Conference on Machine Learning*, pp. 5958–5968. PMLR, 2020b.

Lieber, O., Lenz, B., Bata, H., Cohen, G., Osin, J., Dalmedigos, I., Safahi, E., Meirom, S., Belinkov, Y., Shalev-Shwartz, S., Abend, O., Alon, R., Asida, T., Bergman, A., Glozman, R., Gokhman, M., Manevich, A., Ratner, N., Rozen, N., Shwartz, E., Zusman, M., and Shoham, Y. Jamba: A hybrid transformer-mamba language model, 2024. URL `https://arxiv.org/abs/2403.19887`.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the middle: How language models use long contexts. *CoRR*, abs/2307.03172, 2023a. doi: 10.48550/arXiv.2307.03172. URL `https://doi.org/10.48550/arXiv.2307.03172`.

Liu, Z. L., Dettmers, T., Lin, X. V., Stoyanov, V., and Li, X. Towards a unified view of sparse feed-forward network in pretraining large language model, 2023b.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization, 2019.

Ma, X., Zhou, C., Kong, X., He, J., Gui, L., Neubig, G., May, J., and Zettlemoyer, L. Mega: moving average equipped gated attention. *arXiv preprint arXiv:2209.10655*, 2022.

Maziarz, K., Kokiopoulou, E., Gesmundo, A., Sbaiz, L., Bartok, G., and Berent, J. Gumbel-matrix routing for flexible multi-task learning. *arXiv preprint arXiv:1910.04915*, 2019.

McCulloch, W. S. and Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

Muqeeth, M., Liu, H., and Raffel, C. Soft merging of experts with adaptive routing, 2023.

Mustafa, B., Riquelme, C., Puigcerver, J., Jenatton, R., and Houlsby, N. Multimodal contrastive learning with limoe: the language-image mixture of experts, 2022.

Narang, S., Chung, H. W., Tay, Y., Fedus, W., Fevry, T., Matena, M., Malkan, K., Fiedel, N., Shazeer, N., Lan, Z., Zhou, Y., Li, W., Ding, N., Marcus, J., Roberts, A., and Raffel, C. Do transformer modifications transfer across implementations and applications? *arXiv preprint arXiv:2102.11972*, 2021.

Nvidia. Ampere Architecture, 2020. https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

OpenAI. Gpt-4 technical report, 2023.

OpenAI. Preparedness framework, 2024. URL `https://cdn.openai.com/openai-preparedness-framework-beta.pdf`. Accessed: 2024-09-17.

Orvieto, A., Smith, S. L., Gu, A., Fernando, A., Gulcehre, C., Pascanu, R., and De, S. Resurrecting recurrent neural networks for long sequences. *arXiv preprint arXiv:2303.06349*, 2023.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P. F., Leike, J., and Lowe, R. Training language models to follow instructions with human feedback. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library, 2019.

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Biderman, S., Cao, H., Cheng, X., Chung, M., Grella, M., GV, K. K., He, X., Hou, H., Lin, J., Kazienko, P., Kocon, J., Kong, J., Koptyra, B., Lau, H., Mantri, K. S. I., Mom, F., Saito, A., Song, G., Tang, X., Wang, B., Wind, J. S., Wozniak, S., Zhang, R., Zhang, Z., Zhao, Q., Zhou, P., Zhou, Q., Zhu, J., and Zhu, R.-J. Rwkv: Reinventing rnns for the transformer era, 2023a.

Peng, B., Quesnelle, J., Fan, H., and Shippole, E. Yarn: Efficient context window extension of large language models, 2023b.

Poli, M., Massaroli, S., Nguyen, E., Fu, D. Y., Dao, T., Baccus, S., Bengio, Y., Ermon, S., and Ré, C. Hyena hierarchy: Towards larger convolutional language models, 2023.

Prato, G., Charlaix, E., and Rezagholizadeh, M. Fully quantized transformer for machine translation. *arXiv preprint arXiv:1910.10485*, 2019.

Puigcerver, J., Riquelme, C., Mustafa, B., and Houlsby, N. From sparse to soft mixtures of experts, 2023.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. Improving language understanding by generative pre-training. 2018.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Rae, J. W., Borgeaud, S., Cai, T., Millican, K., Hoffmann, J., Song, F., Aslanides, J., Henderson, S., Ring, R., Young, S., Rutherford, E., Hennigan, T., Menick, J., Cassirer, A., Powell, R., van den Driessche, G., Hendricks, L. A., Rauh, M., Huang, P.-S., Glaese, A., Welbl, J., Dathathri, S., Huang, S., Uesato, J., Mellor, J., Higgins, I., Creswell, A., McAleese, N., Wu, A., Elsen, E., Jayakumar, S., Buchatskaya, E., Budden, D., Sutherland, E., Simonyan, K., Paganini, M., Sifre, L., Martens, L., Li, X. L., Kuncoro, A., Nematzadeh, A., Gribovskaya, E., Donato, D., Lazaridou, A., Mensch, A., Lespiau, J.-B., Tsimpoukelli, M., Grigorev, N., Fritz, D., Sottiaux, T.,

Pajarskas, M., Pohlen, T., Gong, Z., Toyama, D., de Masson d'Autume, C., Li, Y., Terzi, T., Mikulik, V., Babuschkin, I., Clark, A., de Las Casas, D., Guy, A., Jones, C., Bradbury, J., Johnson, M., Hechtman, B., Weidinger, L., Gabriel, I., Isaac, W., Lockhart, E., Osindero, S., Rimell, L., Dyer, C., Vinyals, O., Ayoub, K., Stanway, J., Bennett, L., Hassabis, D., Kavukcuoglu, K., and Irving, G. Scaling language models: Methods, analysis & insights from training gopher, 2022.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, October 2019.

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

Rajbhandari, S., Li, C., Yao, Z., Zhang, M., Aminabadi, R. Y., Awan, A. A., Rasley, J., and He, Y. DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale, 2022. URL `https://proceedings.mlr.press/v162/rajbhandari22a/rajbhandari22a.pdf`.

Raposo, D., Ritter, S., Richards, B., Lillicrap, T., Humphreys, P. C., and Santoro, A. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, April 2024. URL `https://arxiv.org/abs/2404.02258`.

Raschka, S. *Build a Large Language Model (From Scratch).* Manning Publications, Shelter Island, NY, September 2024. ISBN 9781633437166.

Robertson, S. E. and Zaragoza, H. The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, 2009. doi: 10.1561/1500000019.

Roy, A., Saffar, M., Vaswani, A., and Grangier, D. Efficient content-based sparse attention with routing transformers. *arXiv preprint arXiv:2003.05997*, 2020.

Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Ferrer, C. C., Grattafiori, A., Xiong, W., Défossez, A., Copet, J., Azhar, F., Touvron, H., Martin, L., Usunier, N., Scialom, T., and Synnaeve, G. Code llama: Open foundation models for code, 2023.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Schuster, T., Fisch, A., Gupta, J., Dehghani, M., Bahri, D., Tran, V., Tay, Y., and Metzler, D. Confident adaptive language modeling. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 17456–17472. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/6fac9e316a4ae75ea244ddcef1982c71-Paper-Conference.pdf`.

Shaham, U., Segal, E., Ivgi, M., Efrat, A., Yoran, O., Haviv, A., Gupta, A., Xiong, W., Geva, M., Berant, J., and Levy, O. Scrolls: Standardized comparison over long language sequences, 2022.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017.

Shazeer, N., Cheng, Y., Parmar, N., Tran, D., Vaswani, A., Koanantakool, P., Hawkins, P., Lee, H., Hong, M., Young, C., Sepassi, R., and Hechtman, B. Mesh-tensorflow: Deep learning for supercomputers, 2018.

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

Shi, F., Chen, X., Misra, K., Scales, N., Dohan, D., Chi, E. H., Schärli, N., and Zhou, D. Large language models can be easily distracted by irrelevant context. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 31210–31227. PMLR, 2023a. URL `https://proceedings.mlr.press/v202/shi23a.html`.

Shi, W., Min, S., Lomeli, M., Zhou, C., Li, M., James, R., Lin, X. V., Smith, N. A., Zettlemoyer, L., Yih, S., and Lewis, M. In-context pretraining: Language modeling beyond document boundaries, 2023b.

Shumailov, I., Shumaylov, Z., Zhao, Y., Papernot, N., Anderson, R., and Gal, Y. Ai models collapse when trained on recursively generated data. *Nature*, 631:755–759, July 2024. doi: 10.1038/s41586-024-04456-7. URL `https://www.nature.com/articles/s41586-024-04456-7`.

Smith, J. T. H., Warrington, A., and Linderman, S. W. Simplified state space layers for sequence modeling, 2023.

Strubell, E., Ganesh, A., and McCallum, A. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

Su, J., Lu, Y., Pan, S., Wen, B., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *CoRR*, abs/2104.09864, 2021. URL `https://arxiv.org/abs/2104.09864`.

Sukhbaatar, S., Grave, E., Bojanowski, P., and Joulin, A. Adaptive attention span in transformers. *arXiv preprint arXiv:1905.07799*, 2019.

Sun, X., Choi, J., Chen, C.-Y., Wang, N., Venkataramani, S., Srinivasan, V. V., Cui, X., Zhang, W., and Gopalakrishnan, K. Hybrid 8-bit floating point (hfp8) training and inference for deep neural networks. *Advances in neural information processing systems*, 32:4900–4909, 2019.

Sun, Y., Dong, L., Huang, S., Ma, S., Xia, Y., Xue, J., Wang, J., and Wei, F. Retentive network: A successor to transformer for large language models, 2023.

Sun, Z., Yu, H., Song, X., Liu, R., Yang, Y., and Zhou, D. Mobilebert: a compact task-agnostic bert for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2158–2170, 2020.

Sutton, R. The bitter lesson. *Incomplete Ideas (blog)*, 13(1), 2019.

Tay, Y., Bahri, D., Yang, L., Metzler, D., and Juan, D.-C. Sparse sinkhorn attention, 2020a.

Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*, 2020b.

TogetherComputer. Redpajama: An open source recipe to reproduce llama training dataset, April 2023. URL https://github.com/togethercomputer/RedPajama-Data.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models, 2023.

Tunstall, L., von Werra, L., and Wolf, T. *Natural Language Processing with Transformers, Revised Edition*. O'Reilly Media, 1st edition, July 2022. ISBN 978-1098136796.

Turc, I., Chang, M.-W., Lee, K., and Toutanova, K. Well-read students learn better: On the importance of pre-training compact models, 2019.

Tworkowski, S., Staniszewski, K., Pacek, M., Wu, Y., Michalewski, H., and Miłoś, P. Focused transformer: Contrastive training for context scaling, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

Wang, B. Mesh-Transformer-JAX: Model-Parallel Implementation of Transformer Language Model with JAX. https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

Wang, H., Wu, Z., Liu, Z., Cai, H., Zhu, L., Gan, C., and Han, S. Hat: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187*, 2020.

Wu, Y., Rabe, M. N., Hutchins, D., and Szegedy, C. Memorizing transformers. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=TrjbxzRcnf-.

Xin, J., Tang, R., Lee, J., Yu, Y., and Lin, J. DeeBERT: Dynamic early exiting for accelerating BERT inference. In Jurafsky, D., Chai, J., Schluter, N., and Tetreault, J. (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2246–2251, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.204. URL https://aclanthology.org/2020.acl-main.204.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A. C., Salakhutdinov, R., Zemel, R. S., and Bengio, Y. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044, 2015. URL http://arxiv.org/abs/1502.03044.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In Riloff, E., Chiang, D., Hockenmaier, J., and Tsujii, J. (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language*

Processing, Brussels, Belgium, October 31 - November 4, 2018, pp. 2369–2380. Association for Computational Linguistics, 2018. doi: 10.18653/V1/D18-1259. URL https://doi.org/10.18653/v1/d18-1259.

Zaheer, M., Guruganesh, G., Dubey, A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. Big bird: Transformers for longer sequences. *arXiv preprint arXiv:2007.14062*, 2020.

Zhai, S., Talbott, W., Srivastava, N., Huang, C., Goh, H., Zhang, R., and Susskind, J. An attention free transformer, 2021.

Zhang, J., Zhao, Y., Saleh, M., and Liu, P. J. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization, 2020.

Zhang, X., Shen, Y., Huang, Z., Zhou, J., Rong, W., and Xiong, Z. Mixture of attention heads: Selecting attention heads per token. *arXiv preprint arXiv:2210.05144*, 2022. URL https://arxiv.org/abs/2210.05144.

Zhao, Y., Gu, A., Varma, R., Luo, L., Huang, C.-C., Xu, M., Wright, L., Shojanazeri, H., Ott, M., Shleifer, S., Desmaison, A., Balioglu, C., Damania, P., Nguyen, B., Chauhan, G., Hao, Y., Mathews, A., and Li, S. Pytorch fsdp: Experiences on scaling fully sharded data parallel, 2023.

Zhao, Y., Qu, Y., Staniszewski, K., Tworkowski, S., Liu, W., Miłoś, P., Wu, Y., and Minervini, P. Analysing the impact of sequence composition on language model pre-training, 2024.

Zhou, A., Ma, Y., Zhu, J., Liu, J., Zhang, Z., Yuan, K., Sun, W., and Li, H. Learning n:m fine-grained structured sparse neural networks from scratch. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=K9bw7vqp_s.

Zhou, Y., Lei, T., Liu, H., Du, N., Huang, Y., Zhao, V., Dai, A., Chen, Z., Le, Q., and Laudon, J. Mixture-of-experts with expert choice routing, 2022.

Zhou, Y., Du, N., Huang, Y., Peng, D., Lan, C., Huang, D., Shakeri, S., So, D., Dai, A., Lu, Y., Chen, Z., Le, Q., Cui, C., Laundon, J., and Dean, J. Brainformers: Trading simplicity for efficiency, 2023.

Zhu, L., Liao, B., Zhang, Q., Wang, X., Liu, W., and Wang, X. Vision mamba: Efficient visual representation learning with bidirectional state space model, 2024.

Zohourianshahzadi, Z. and Kalita, J. K. Neural attention for image captioning: review of outstanding methods. *Artificial Intelligence Review*, 55(5):3833–3862, November 2021. ISSN 1573-7462. doi: 10.1007/s10462-021-10092-2. URL http://dx.doi.org/10.1007/s10462-021-10092-2.

Zoph, B., Bello, I., Kumar, S., Du, N., Huang, Y., Dean, J., Shazeer, N., and Fedus, W. St-moe: Designing stable and transferable sparse expert models, 2022.