TOWARDS A UNIFIED VIEW OF SPARSE FEED-FORWARD NETWORK IN TRANSFORMER

Anonymous authors
Paper under double-blind review

ABSTRACT

Large and sparse feed-forward networks (S-FFN) such as Mixture-of-Experts (MoE) have demonstrated to be an efficient approach for scaling up Transformers model size for *pretraining*. By only activating part of the FFN parameters conditioning on input, S-FFN improves generalization performance while keeping training and inference cost (in FLOPs) fixed. A growing body of work has been focusing on improving the S-FFN design, including routing and load balancing methods in the context of MoEs. Previously, another line of work motivates from a neural memory perspective and develops sparse neural memory techniques for S-FFN. This work merges the two seemingly different lines of work. We present a unified framework to categorize design choices along two axes: memory block size and memory block selection method. Using this unified framework, we compare several S-FFN architectures for language modeling and provide insights into their relative efficacy and efficiency. We show that a smaller memory block size leads to lower perplexity. Additionally, we find that selection through a gate, in general, improves the perplexity-FLOPs trade-off but has worse perplexity than selection using hidden states without a gate. Based our analysis, we propose a new selection method — **`Avg-K`** that selects blocks through their mean aggregated hidden states. Not only outperforming all MoE models considered, **`Avg-K`** is also the first to performs well without load balancing while Switch Transformer may degenerate.

1 INTRODUCTION

Large-scale pretrained language models (LLMs) achieve remarkable performance and generalization ability for NLP tasks (Radford & Narasimhan, 2018; Devlin et al., 2019; Liu et al., 2019; Radford et al., 2019; Brown et al., 2020; Raffel et al., 2020; Chowdhery et al., 2022). Scaling up the model size (the number of parameters) has been shown as a reliable recipe for better generalization, unlocking new capabilities, while the performance has not shown signs of plateauing (Kaplan et al., 2020; Zhang et al., 2022a; Chowdhery et al., 2022; Hoffmann et al., 2022; Wei et al., 2022). However, the computational resources required to train larger language models are formidable, calling for more efficient training and inference solutions of LLMs (Borgeaud et al., 2022; Schwartz et al., 2020; Tay et al., 2020).

One promising direction is sparse scaling which increases the number of parameters while keeping the training and inference cost (in FLOPs) fixed. Recent work focuses on scaling up a transformer's feed-forward network (FFN) with sparsely activated parameters, resulting in a scaled and sparse FFN (S-FFN). There have been two major approaches to achieve S-FFN. One treats S-FFN as a neural memory (Sukhbaatar et al., 2015a) where a sparse memory retrieves and activates only parts of the memory cells (Lample et al., 2019). The other adopts Mixture-of-Expert Network (MoE) (Lepikhin et al., 2021; Fedus et al., 2021; Du et al., 2021; Roller et al., 2021; Lewis et al., 2021; Chi et al., 2022) that replaces a single FFN module with multiple equal-sized ones (called "experts") and only activates a few among many experts for a particular input.

While both memory and MoE models achieve S-FFN, they have been considered two completely different approaches. We aim to draw the connections between these two classes of S-FFN: What critical design choices do they have in common? Which design choices are essential for their modeling capability and computation efficiency? Can the effective ingredients of each method be transferred and combined to improve performance further?

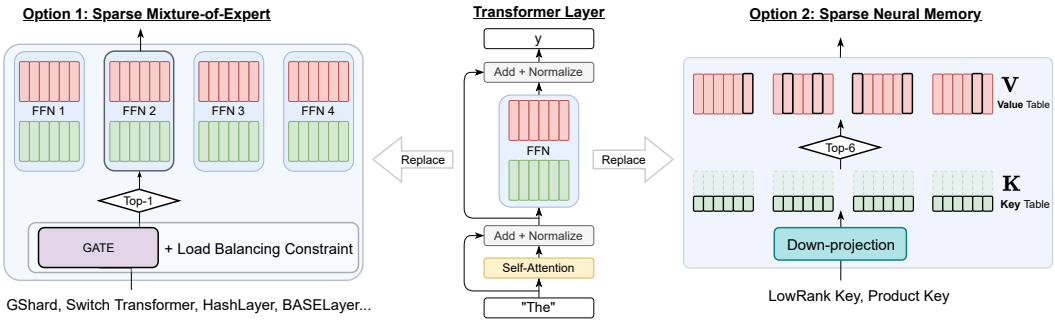

Figure 1: Sparse Mixture-of-Expert and Sparse Neural Memory as two different methods.

In order to answer these questions, we start from the neural memory view of FFN (Sukhbaatar et al., 2015a) (§2.1) and reduce all S-FFN to the same mathematical form (§3.1). Then, we characterize various S-FFN methods along two dimensions — memory block size (e.g. expert size) and memory block selection method (e.g. gating) (§3.2).

Using this framework, we made the following contributions:

- We study a wide range of memory block sizes besides common block size in MoEs (Fedus et al., 2022) and show that, compared with a larger one, a smaller block size can keep improving the perplexity with little incurred extra FLOPs (§5.1 §5.2), leading to better perplexity/computation trade-offs.
- We conduct a systematic exploration of block selection methods to quantify their relative efficacy and efficiency (§5.2). Specifically, we find that the selection method through a gating function, in general, improves the FLOPs-Perplexity trade-off. However, the parameterization of the current gating function has worse perplexity than a selection method that uses the FFN hidden states.
- Drawing on insights above, we propose a simple gate for S-FFN— **`Avg-K`** (§6), which efficiently selects memory blocks based on the mean aggregated hidden states of each block. With 1% additional FLOPs, **`Avg-K`** achieves 2.16 lower perplexity than a vanilla transformer (16.96), outperforming Switch Transformer (16.45). **`Avg-K`** is the first MoE model performs well without load balancing constraint, as suggested by previous work (Shazeer et al., 2017; Eigen et al., 2014).

2 Background

2.1 Feed-Forward Network

A transformer layer (Vaswani et al., 2017) consists of a self-attention block and a Feed-Forward Network (FFN) block. FFN receives an input vector $\mathbf{x} \in \mathbb{R}^d$ from the self-attention block, multiplies it with $\mathbf{K} \in \mathbb{R}^{d_m \times d}$, applies an non-linear function to obtain the hidden states $\mathbf{m} \in \mathbb{R}^{d_m}$ and applies another affine transformation $\mathbf{V} \in \mathbb{R}^{d_m \times d}$ to produce a d-dimensional output (Eq. 1).

$$\text{FFN}(\mathbf{x}) = f(\mathbf{x} \cdot \mathbf{K}^\top) \cdot \mathbf{V} = \mathbf{m} \cdot \mathbf{V} \in \mathbb{R}^d \quad \text{(Multi-Layer Perceptron view)} \tag{1}$$

$$\text{FFN}(\mathbf{x}) = \sum_{i=0}^{d_m-1} f(\mathbf{x} \cdot \mathbf{k}_i) \cdot \mathbf{v}_i = \sum_{i=0}^{d_m-1} m_i \cdot \mathbf{v}_i \in \mathbb{R}^d \quad \text{(Neural Memory view)} \tag{2}$$

FFN is a multi-layer perceptron, but it can also be described as a neural memory (Sukhbaatar et al., 2015b; 2019; Geva et al., 2021)(Eq. 2). In this view, FFN consists of d_m key-value pairs, called *memory cells*. Each key is represented by a d-dimensional $\mathbf{k}_i \in \mathbb{R}^d$, and together form the *key table*; likewise, a *value table* $\mathbf{V} \in \mathbb{R}^{d_m \times d}$. When the memory receives the query input $\mathbf{x} \in \mathbb{R}^d$, it multiplies $\mathbf{x}$ with every $\mathbf{k}_i$; followed by the non-linear function, it produces *memory coefficient* $m_i = f(\mathbf{x} \cdot \mathbf{k}_i)$ for the i-th memory cell. Finally, the output of FFN is the sum of its values $\mathbf{v}_i$ weighted by their corresponding memory coefficient m_i. Conventionally, the *size* of FFN — d_m — is set to be $4 \cdot d$.

2.2 SCALING UP FFN

As discussed in §1, scaling up the number of parameters in FFN serves as a lever to improve transformer performance. Since a conventional FFN already takes about two-thirds of a transformer layer's parameters (Geva et al., 2021), scaling up FFN will greatly affect the parameter size of a transformer model. However, one could sparsely activate the parameter to control the required compute. In this section, we will review two lines of work to achieve a *scaled* and *sparse* FFN (S-FFN). One has a mixture-of-expert model activate a few experts (§2.2.1), and the other specifies a memory model (§2.2.2).

2.2.1 MIXTURE OF EXPERTS (MOE)

Mixture of experts (MoE; Jacobs et al. (1991)) consists of a set of expert models $\{\mathbf{f}_i(\mathbf{x})\}_{i=0}^{B-1}$ and a gating function $\mathbf{g} : \mathbb{R}^d \to \mathbb{R}^B$ to estimates the importance of each expert. Finally, the output is the sum of experts' output weighted by the gate's weight estimation for that particular expert.

$$\text{MoE}(\mathbf{x}) = \sum_{i \in \mathcal{E} = \{0,1,\cdots,B-1\}} \mathbf{g}_i(\mathbf{x}) \cdot \mathbf{f}_i(\mathbf{x}) \tag{3}$$

Recent work (Du et al., 2021; Lepikhin et al., 2021; Roller et al., 2021; Lewis et al., 2021; Zhou et al., 2022) adopt it to transformer by treating an FFN as one expert and **S**parsely activating the **MoE** (**SMoE**). In SMoE, the gating function (or "router") routes an input token $\mathbf{x}$ to a subset (e.g. 1 or 2) of experts — $\mathcal{E} = \text{subset}(\mathbf{g}(\mathbf{x}))$. Conventionally, SMoE enforces **load balancing** constraints to avoid overly using a few experts while under-utilizing others, and converging to local optima (Shazeer et al., 2017; Eigen et al., 2014). Various SMoEs mainly have two types of gates.

Learned gate is parameterized by a set of learnable expert embeddings $\boldsymbol{\theta} = [\mathbf{e}_0; \cdots; \mathbf{e}_{B-1}] \in \mathbb{R}^{B \times d}$, corresponding to each experts. The importance of the i-th expert is obtained by $\mathbf{g}_i(\mathbf{x}) = \frac{\exp(\mathbf{e}_i \cdot \mathbf{x})}{\sum_j \exp(\mathbf{e}_j \cdot \mathbf{x})}$. To enforce load balancing when routing, SMoEs use an additional auxiliary loss (Lepikhin et al., 2021; Fedus et al., 2021; Artetxe et al., 2021; Du et al., 2021; Chi et al., 2022) or frame expert utilization as a constrained optimization problem (Lewis et al., 2021; Zhou et al., 2022).

Static gate, in contrast to a learnable gate, does not have any differentiable parameters. Instead, it uses a static mapping that encodes load-balancing constraints to route input (Roller et al., 2021; Gururangan et al., 2021). For example, RandHash from HashLayer (Roller et al., 2021) uses a hash table that maps from token type to *randomly* selected expert(s) and enforces the load balancing with random selection. DEMix (Gururangan et al., 2021) ensures i-th expert only sees data from i-th domain and trains each expert for equal steps to ensure equal utilization.

2.2.2 SPARSE NEURAL MEMORY

The other line of work follows the memory view of FFN (Eq. 2). It is straightforward to increase the memory size d_m to a much larger value $d_m \gg 4 \cdot d$. By only using the top-k entries in $\mathbf{m}$, one could sparsely activate the value table, resulting in a vanilla sparse memory (**VanillaM**). The straightforward application, however, uses the entire key table to produce all the memory coefficients $\mathbf{m} = \mathbf{x} \cdot \mathbf{K}^\top$, resulting in dense computation proportional to the memory size. Memory models need to use models that do not look at the full key table to scale computation sublinearly to the number of memory cells and take advantage of sparse computation. Lample et al. (2019) explored the following two techniques in this direction.

Low-Rank Key Memory (LoRKM) A straightforward technique is to assume that the key table is composed of and approximated by a down-projection $\mathbf{D} \in \mathbb{R}^{d \times d_\ell}$ and a low-rank key table $\tilde{\mathbf{K}} \in \mathbb{R}^{d_m \times d_\ell}$, where $d_\ell \ll d$. LoRKM produces memory coefficients by $m_i = f((\mathbf{x} \cdot \mathbf{D}) \cdot \tilde{\mathbf{k}}_i) = f(\mathbf{t} \cdot \tilde{\mathbf{k}}_i)$.

Product Key Memory (PKM) Building upon LoRKM, PKM assumes that the low-rank key table is further approximated by the outer *product*, w.r.t. vector concatenation operator, of two smaller *sub-key* tables $\mathbf{C}, \mathbf{C}' \in \mathbb{R}^{\sqrt{d_m} \times \frac{d_\ell}{2}}$. Unlike LoRKM where key vectors are independent of each other, key vectors in PKM have some overlaps with each other, i.e. $\tilde{\mathbf{k}}_i = \left[\mathbf{c}_{\lfloor i/\sqrt{d_m} \rfloor}, \mathbf{c}'_{i \pmod{\sqrt{d_m}}}\right] \in \mathbb{R}^{d_\ell}$. Due to such factorization, PKM has a negligible key table $\tilde{\mathbf{K}} \cdot \mathbf{D}^\top$ (e.g., $< 0.3\%$) relative to the parameters in the value table.

3 A UNIFIED VIEW OF SPARSE FFNS

We show the connections between MoE and neural memory despite their different surface forms. We first derive an equivalent form of MoE to establish its connection with sparse memory (§3.1). Then, we propose a unified framework for S-FFN (§3.2).

3.1 A CLOSER LOOK AT MOE

MoEs use a gating function to estimate the importance of all experts and combine each expert's output through linear combination. Here, inspired by the memory view on FFNs (§2), we derive a formulation of MoE models (Eq. 3) as seen from the memory view:

$$\begin{aligned}
\text{MoE}(\mathbf{x}) &= \sum_{i=0}^{B-1} \mathbf{g}_i(\mathbf{x}) \cdot \mathbf{f}_i(\mathbf{x}) = \sum_{i=0}^{B-1} \mathbf{g}_i(\mathbf{x}) \cdot \text{FFN}^{(i)}(\mathbf{x}) \quad \text{(Each expert is an FFN)} \\
&= \sum_{i=0}^{B-1} \mathbf{g}_i(\mathbf{x}) \cdot \left(\sum_{j=0}^{d_m-1} m_j^{(i)} \cdot \mathbf{v}_j^{(i)} \right) \quad \text{(Apply the memory view of FFN (Eq. 2))} \\
&= \sum_{i=0}^{B-1} \sum_{j=0}^{d_m-1} \left(\mathbf{g}_i(\mathbf{x}) \cdot m_j^{(i)} \right) \cdot \mathbf{v}_j^{(i)} \quad \text{(Distributive property of multiplication)} \qquad (4) \\
&= \sum_{l=0 \text{ s.t. } l=i\cdot d_m+j}^{B\cdot d_m-1} \left(\mathbf{g}_i(\mathbf{x}) \cdot m_j^{(i)} \right) \cdot \mathbf{v}_j^{(i)} \text{ (Re-indexing)} = \sum_{l=0}^{B\cdot d_m-1} m_l \cdot \mathbf{v}_l \qquad (5)
\end{aligned}$$

In Eq 5, $\mathbf{v}_l$ is the l-th row of the stack of d_m-size value tables from B FFN experts; and $m_l = m_{i\cdot d_m+j}$ is its memory coefficient obtained from $\mathbf{g}_i(\mathbf{x}) \cdot m_j^{(i)}$. For example, m_{d_m} is is obtained by weighting $m_0^{(1)}$ — the 0-th memory coefficient in the 1-th FFN expert — with $\mathbf{g}_0(\mathbf{x})$ — the estimation of the 1-th FFN from gate. Building upon such memory view, one could see that is a sparse memory operating in terms block and uses its gate to narrow down the summation over the stacked value tables $\mathbf{V}$ to ones in $\text{FFN}^{(i)}$, for $i \in \text{subset}(\mathbf{g}(\mathbf{x})) = \mathcal{E}$.

Comparison with Sparse Memory Both SMoE and Sparse Memory are neural memory, but there are several differences: **1)** Memory cells share the same importance weight: in sparse memory, each memory cell receives an individual weight. In contrast, in SMoE groups of $4 \cdot d$ share the same importance weight $\mathbf{g}_i(\mathbf{x})$. **2)** Single selection and importance weight: Sparse memory uses a single dot product between input token and any key vector $\mathbf{x} \cdot \mathbf{k}$ for both, whereas SMoE depends on a separately parameterized gate $\mathbf{g}$ in addition to the dot product.

3.2 THE UNIFIED FRAMEWORK

We propose a general framework that unifies the two different approaches to achieve S-FFN. We cast both as instances of a memory with large key and value table — $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{d_m \times d}$, where $d_m \gg 4 \cdot d$. We distinguish the different methods along two dimensions illustrated below and summarized in Table 1:

Memory block size specifies how many memory cells share the same importance weight at selection time, and thus together treated as a memory *block*. We use g to denote the size of one block. In other words, we split the $\mathbf{K}, \mathbf{V}$ along the d_m-dimension into g-size blocks. Therefore, a memory consists of $B = d_m/g$ blocks in total. Formally, we write

$$\mathbf{K}^g = \left[\mathbf{K}^{(0)}; \mathbf{K}^{(1)}; \cdots; \mathbf{K}^{(B-1)} \right] \in \mathbb{R}^{d_m \times d_k}, \mathbf{V}^g = \left[\mathbf{V}^{(0)}; \mathbf{V}^{(1)}; \cdots; \mathbf{V}^{(B-1)} \right] \in \mathbb{R}^{d_m \times d_v}$$

For example, sparse memory has block size $g = 1$ — trivially treating 1 memory cell as a "block"; and SMoE has the block size $g = 4 \cdot d$ (§3.1). Current approaches generally use fixed block sizes, but this is mostly an artifact of how the methods were derived rather than a mathematical constraint. For example, we can design SMoE versions instead of 1 expert of size $4 \cdot d$, or uses 2 experts of size $2 \cdot d$. We can similarly chunk memory coefficients $\mathbf{m}$ into blocks of size g in sparse memories.

Table 1: S-FFN methods decomposed along the defined design dimensions.

Memory block size (g)	**Memory block selection method**		**Model Name**
1	Direct	Full-parameter Key	VanillaM
		Low-rank Key	LoRKM, PKM (Lample et al., 2019)
$4 \cdot d$	Indirect	Learned gate	Switch Transformer(Fedus et al., 2021), GShard (Lepikhin et al., 2021), GLaM (Du et al., 2021), BASELayer (Lewis et al., 2021), X-MoE (Chi et al., 2022)
		Static gate	HashLayer(Roller et al., 2021), DEMix (Gururangan et al., 2021)

Memory block selection method is the specific function that compute the importance of each memory blocks for selection. Since SMoE is also a type of sparse memory, we distinguish the selection method by a new criterion — *whether one allows input* $\mathbf{x}$ *to directly interact with the key table* $\mathbf{K}^g$. As discussed in §3.1, SMoE uses the estimation from an individually parameterized gate to select, while sparse memory solely and directly uses a key table. Thus, current SMoE is a type of *indirect* selection method, and sparse memory a *direct* one. Various SMoEs are further characterized by whether their gating function has learned parameters or consists of a static mapping (§2.2.1). Meanwhile, sparse memory is characterized by how much factorization the key table uses (§2.2.2).

4 Experiment Setup

4.1 Models

We choose **Dense Baseline** using transformer architectures used in GPT-3 models (Brown et al., 2020), which has 24 transformer layers, with $d = 1024$, GeLU activation functions and with a memory size (or FFN hidden size) to be $4 \cdot d$.

S-FFN Given a model above, we replace some of its FFNs with an S-FFN. Similar to (Lepikhin et al., 2021), we replace the FFN at every 6 layers, leading to 4 S-FFNs in total across 24 layers. Since memory block size g is a perception imposed upon memory cells, we use k to denote the number of active memory cells and control how activated the S-FFN is. We use the formulation of $d_m = E \cdot (4 \cdot d)$ to control the size of S-FFN, so the S-FFN will activate $b = \frac{k}{g}$ out of $B = \frac{d_m}{g}$ memory blocks. In Table 2, we list all S-FFN models used for analysis in §5. We count FLOPs analytically following Narayanan et al. (2021) and do not account if a worker finishes computation before another (when using model parallelism). We use *the number of learnable parameter* to consider whether two models are equally expressive.

PKM-FFN Since the factorized key table in PKM has little ($< 0.3\%$) learnable parameter relative to the value table, we propose a variant called **PKM-FFN** to match the number of parameter of other models like RandHash. This variant has memory block size $g = 1$ and the same key-value table as RandHash. PKM-FFN has a gate whose $\mathbf{g}(\mathbf{x})$ is the same as the $\mathbf{m}$ from a PKM and $\mathbf{g}_i = m_i$.

4.2 Language Modeling

Pretraining Data We pretrain all S-FFN models on a total of 453GB text with 112B tokens from a union of six English-only datasets, including English subset of CC100 (Wenzek et al., 2020) and the five datasets used to pretrain RoBERTa (Liu et al., 2019) — specifically BookCorpus (Zhu et al., 2015), English Wikipedia, CC-News (Nagel, 2016), OpenWebText (Gokaslan & Cohen, 2019), CC-Stories (Trinh & Le, 2018) (details in Appendix A.2) We adopt the same Byte-Pair Encoding as GPT-2 (Radford et al., 2019) and RoBERTa (Liu et al., 2019) with a vocabulary of 50K subword units. All models are trained for 60B tokens for convergence. See Appendix A.1 for hyperparameters.

Table 2: All the S-FFN models used in experiments and analysis in §5 — g is the number of memory cells grouped in a memory block, k is the active memory cells, and E control the sizes of a memory $d_m = E \cdot (4 \cdot d)$. Some settings(*) are only used for PKM.

Selection method type	Method name	g	E	k
Direct	VanillaM	$\{1, 64, 256, 1024, 2048, 4096\}$ (§5.1)	$\{4, 16, (32)^*\}$	$\{4096, (8192)^*\}$
	LoRKM	$\{1\}$		
	PKM (Lample et al., 2019)	$\{1\}$		
Indirect	RandHash (Roller et al., 2021)	$\{1, 64, 256, 1024, 2048, 4096\}$ (§5.1)		
	Switch (Fedus et al., 2021)	$\{4096\}$		
	PKM-FFN (§4.1)	$\{1\}$		

Evaluation settings We evaluate our models' ability to predict the next token in a sequence as measured by perplexity. We report both **in-domain** and **out-of-domain** perplexity to indicate generalization ability. For out-of-domain, we use data from The Pile (Gao et al., 2020), a public dataset that combines data from 22 diverse sources.

5 Analysis under the unified view

In this section, we use the proposed unified view to systematically study the design choice of S-FFN. Specifically, **(1)**. we study a wide range of block sizes other than the incidental choice used in existing work and investigate its impact on language modeling perplexity (§5.1). **(2)**. Both direct and indirect block selection methods lead to lower perplexity than a standard FFN, but which type of method has better FLOPs-Perplexity trade-off and what are the relative efficacy and efficiency of different methods require further study (§5.2).

5.1 Memory block size

Since block size is a natural number, we aim to answer a straightforward question — **given a fixed k, does smaller memory block size lead to lower perplexity?** We use simple and robust selection methods to disentangle the impact of hyperparameter choices. Specifically, we use random hash in HashLayer (Roller et al., 2021) (denoted **RandHash**) for indirect block selection and exact top-k memory block (denoted **VanillaM**) for direct block selection. For all experiments, we use $E = 16$.

RandHash randomly selects $b = \frac{k}{g}$ unique memory blocks among all $B = \frac{d_m}{g}$ blocks — essentially sampling b unique values from Uniform($[0, \cdots, B-1]$). Originally, with block size $g = 4096$, a RandHash assigns a token to $\frac{4096}{4096} = 1$ block; with block size $g = 2048$, $\frac{4096}{2048} = 2$ blocks.

VanillaM originally has a block size $g = 1$ and selects top-k scalars in memory coefficients $\mathbf{m} = \text{GeLU}(\mathbf{x} \cdot \mathbf{K}^\top)$. We made a minimal change to extend it to larger block size g: given $\mathbf{m}$, we chunk it into B blocks — $\mathbf{m}^g = [\mathbf{m}^{(0)}; \cdots; \mathbf{m}^{(B-1)}]$; then, we select the top-b blocks using the average of each block — $\texttt{Avg}(\text{GeLU}(\mathbf{x} \cdot (\mathbf{K}^{(i)})^\top), \texttt{dim=0})$.[1]

In Fig. 2, we observe that smaller block size leads to an improvement of $0.4 (15.75 \rightarrow 15.35)$ perplexity for RandHash and an improvement of $0.87 (15.56 \rightarrow 14.69)$ for VanillaM. This observation holds no matter how one selects: **1)** with indirect (RandHash) or direct (VanillaM) selection and **2)** with random (RandHash) or similarity-based (VanillaM) block selection. In Appendix B.2, we provide theoretical justifications for this observation which shows that a smaller block size improves model capacity by including more combinations of memory cells. For example, with $g/2$, half memory cells of expert-1 could be activated together with half of the expert-2; however, this combination is impossible with larger block size.

Costs for a smaller block size g With model parallelism (Lepikhin et al., 2021), multiple GPUs contains different memory block and parallelize the calculations. If with block size $g = 4 \cdot d$, a

[1] $\texttt{Avg}(\cdot)$ performs better than other simple aggregators — $\texttt{Min}(\cdot)$, $\texttt{Max}(\cdot)$, and $\texttt{Avg}(|\cdot|)$; see Table 7

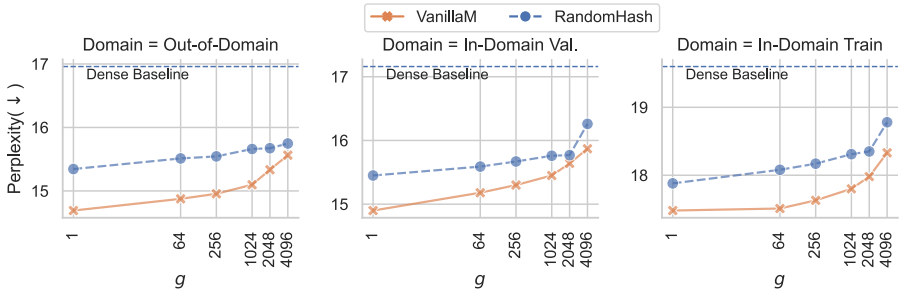

Figure 2: Perplexity (lower the better) consistently improve as memory block size g decreases for both direct (VanillaM) and indirect (RandHash) selection method in S-FFN models.

token is only routed to 1 memory block on one device, each device doubles its chance to receive more tokens with block size $g = 2 \cdot d$. Therefore, each GPU processes more tokens and requires more computation time. Better implementations could be developed to make a smaller block size for practical usage. In addition, since each memory block has its representation stored in the gating function, the smaller block will lead to more block representation stored in the gate, e.g., more learned parameters in the learned gate and a larger table for the static gate. Although RandHash with memory block size $g = 4 \cdot d$ cost essentially the same with memory block size $g = 1$, computing $\mathbf{g}(\mathbf{x})$ for learned gates requires more cost (details in Appendix B.3.1).

5.2 Block Selection Method

Next, we investigate the impact of the selection method, specifically, the FLOPs-perplexity trade-off for direct and indirect methods to determine the overall usefulness of each S-FFN method.

FLOPs-perplexity trade-off We study the efficiency of direct and indirect selection methods in S-FFN models characterized by FLOPS-perplexity trade-off. We conduct experiments across different scales of the memory by varying $E \in \{4, 16\}$; additionally, we run $E = 32$ for PKM.

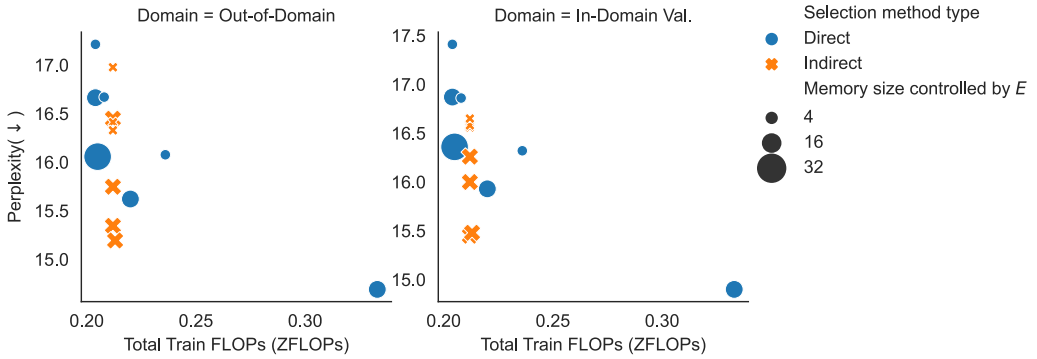

Figure 3: FLOPs-perplexity trade-off of indirect block selection is better than that of direct block selection (see Fig. 5 for a detailed plot). Indirect methods (orange cross) have more perplexity improvement relative to increases in FLOPs than direct methods (blue dots).

In Fig. 3, we marginalize different factors used in the two selection methods — i.e. types of gates, factorization techniques on key table, etc. — and consider each type of selection method as a whole. When we change different marginalized factors, we observe that indirect methods tend to improve more as we use more FLOPs (with larger memory sizes controlled by E). Thus, the indirect method has a better FLOPs-perplexity trade-off.

Effect of gating function We start with contrastive comparisons among PKM-FFN $^{E=16}$, PKM$^{E=32}$, RandHash$^{E=16}$ with memory block size $g = 1$ and 4096 active memory blocks. From the three parameter-matched models, we can learn important lessons to improve the design of gate:

Table 3: List of experiments for contrastively comparing designs. This table assume each memory cell is a memory block, i.e. $g = 1$. The top two best performing models (bolded) have full-parameter key table and depend more on dot product to activate parameters.

Selection method type	Name	# Active memory cells (k)	#Parameters (Entire Model)	Train ZFLOPs	Out-of-Domain Avg. (↓) (See Table 9 for each domain)	In-Domain (↓)	
						Train	Val.
	Dense Baseline	4096	354.7M	0.212	16.96	19.60	17.16
Direct	PKM$^{E=16}$	4096	590.2M	0.205	16.66	19.45	16.87
	PKM$^{E=32}$	4096	858.7M	0.205	16.06	18.93	16.36
	PKM$^{E=32}$	8192	858.7M	0.213	16.16	19.05	16.45
	VanillaM$^{E=16}$	4096	858.3M	0.333	**14.69**	**17.48**	**14.90**
Indirect	PKM-FFN$^{E=16}$	4096	858.9M	0.213	**15.19**	**17.82**	**15.48**
	RandHash$^{E=16}$	4096	858.3M	0.212	15.35	17.88	15.45

1. Comparing with PKM-FFN$^{E=16}$, PKM$^{E=32}$ essentially moves the parameters from a full-paramter key table to double the size of value table.
2. PKM-FFN$^{E=16}$ and RandHash$^{E=16}$ have the same key and value tables; but the former uses gate jointly learned with key table, while the later uses a learning-free gate.

As shown in Table 3, on out-of-domain, PKM-FFN$^{E=16}$ outperforms PKM$^{E=32}$(16.06) by 0.87 perplexity and slightly outperform RandHash$^{E=16}$ by 0.16. Therefore, it is essential to have **a full-parameter, and thus expressive enough, key table** to produce memory coefficients.

Table 3 shows the improvement of VanillaM$^{E=16}$, PKM-FFN$^{E=16}$, RandHash$^{E=16}$ over Dense Baseline (16.96) are 2.27, 1.77, and 1.61 respectively on out-of-domain. They only differ by how much they depends on key table for selection — VanillaM directly uses it, PKM-FFN indirectly gain information from it, and RandHash completely ignores it. Therefore, we conclude that **the more dependent on key table the selection method is, the better** language model it will lead to; and indirect usage (PKM-FFN) is not enough.

6 A NEW ROUTING METHOD — **Avg-K**

Shown by Fig. 3, a gated MoE has clear advantage over sparse memory. Basing on the contrastive analysis from §5.2, we believe the current MoE already has a good design choice — a full-parameter key table. However, an important improvement is to make more use of each expert's key table for routing tokens. Additionally, it needs to be compute-efficient with smaller g (§5.1).

To this end, we propose a new routing method — **Avg-K**. We represent each block with the average of its key table $\mathbf{K}^{(i)} \in \mathbb{R}^{g \times d}$ along g-dimension — $\mathbf{e}_i = \frac{1}{g}\sum_{j=0}^{g-1} \mathbf{k}_j^{(i)} = \mathbf{Avg}(\mathbf{K}^{(i)}, \texttt{dim=0})$. Then, we use the dot product between $\mathbf{x}$ and the averages to select the top-b selected block and route the token there for memory calculation (Eq. 2):

$$\mathbf{g}_i(\mathbf{x}) = \begin{cases} 1 & i \in \mathcal{E} = \{\text{top-}b \text{ of } [\mathbf{e}_0 \cdot \mathbf{x}, \cdots, \mathbf{e}_{B-1} \cdot \mathbf{x}]\} \\ 0 & \text{otherwise} \end{cases}$$

Due to the linearity of averaging, the operation $\mathbf{e}_i \cdot \mathbf{x}$ is equivalent to calculating the average of dot products within a block without GeLU. Since all tokens share the averages, our method is efficient. We provide more rationale for our choice of average function in Appendix C.1.

Table 4 shows that the proposed S-FFN design outperforms all other indirect methods. With $< 1\%$ additional FLOPs, **Avg-K** achieves 2.16 lower perplexity than Dense Baseline (16.96), outperform (Fedus et al., 2021) by 1.65 and (Roller et al., 2021) by 0.5. In Appendix C.2, we include a contrastive analysis with VanillaM to understand the drastic improvement from block size $g = 4096$ to 256.

Table 4: **Avg-K** out-performs other indirect block selection methods.

E	#Parameters (Entire Model)	Selection method	g	Train ZFLOPs	Out-of-Domain Avg. (↓) (See Table 10 for details)	In-Domain (↓) Train	In-Domain (↓) Val.
1	354.7M	Dense Baseline	1	0.212	16.96	19.60	17.16
16	≈ 858.3M	RandHash (Roller et al., 2021)	4096	0.212	15.75	18.78	16.26
			1	0.212	15.35	17.88	15.45
		Switch (Fedus et al., 2021)	4096	0.212	16.45	18.20	16.00
		PKM-FFN	1	0.213	15.19	17.82	15.48
		Avg-K	4096	0.212	16.44	19.04	16.59
			256	0.213	14.91	17.57	15.19
			64	0.214	**14.80**	**17.51**	**15.11**

Load balancing Previously, load balancing has two-fold implication: **1)** to improve empirical training time Fedus et al. (2021); **2)** more importantly, to prevent the expert representations from degeneration — i.e. routing most tokens to a single expert (Shazeer et al., 2017; Eigen et al., 2014). Based on our observation, we hypothesize that load balancing is only *optional* for *learning* a good expert representation (i.e., the second implication) since our experiment with Switch enforces load balancing and experiment with **Avg-K** doesn't. We provide analysis for block usage for **Avg-K** in Appendix C.3.

Limitation Since **Avg-K** essentially applies an average pooling to the key table $\mathbf{K}^g$, a better alternative may exist. Our method also heavily depends on dot product information, but this might not be the best information to be used. Additionally, in large-scale SMoE training, the speed is limited by the most heavy-loaded GPU when model parallelism is used. Therefore, load balancing is essential. We note that our scale is relatively small and does not use model parallelism, so the problem is not pronounced for us. Future follow-up should look at how to incorporate load balancing into the unified framework. Such unification requires more advanced theoretical connection with memory block and block selection method, which likely involves consideration of training procedure.

7 Related Work

Excluded S-FFN Terraformer's Jaszczur et al. (2021) technique on FFN is closest to our PKM-FFN because there is a low-rank learned gate to operate on each memory cells for selection. However, we exclude this method because our framework uses all memory cells in each block, but Terraformer selects 1 cell in each memory block (see our study in Appendix D.1). In finetuning scenario, Zhang et al. (2022b) studies the connection between FFN and SMoE by turning *trained* FFN into experts and separately learning a gate. In contrast, we focus on pretraining *from scratch*.

Approximate Nearest Neighbour(ANN) search One might wonder whether ANN techniques could help to search for the best key in VanillaM rather than trade the expressiveness of the key table for efficiency. For example, one could process the unfactorized key table by ANN methods like FAISS (Johnson et al., 2021) and ScaNN (Guo et al., 2020). One successful example is applying vanilla Locality-Sensitive Hashing to Reformer (Kitaev et al., 2020). However, in our preliminary study, we found that perplexity is greatly affected by the search quality, and building a data structure after every update is expensive and hard to avoid. We leave detailed discussion and analysis to Appendix D.2.

8 Conclusion

We provide a unified framework for designing sparse FFN in transformers and analyze existing S-FFN methods such as SMoEs in the language modeling task. Using this framework, we found that smaller memory block (e.g. expert) size improves perplexity at the cost of slightly higher computation cost. Selection methods with gates have better FLOPs-Perplexity trade-offs than without, while the gating function in current SMoEs is sub-optimal. This framework enables us to instantiate a simpler S-FFN architecture that outperforms SMoEs while still being efficient in training and inference.

REFERENCES

Mikel Artetxe, Shruti Bhosale, Naman Goyal, Todor Mihaylov, Myle Ott, Sam Shleifer, Xi Victoria Lin, Jingfei Du, Srinivasan Iyer, Ramakanth Pasunuru, Giri Anantharaman, Xian Li, Shuohui Chen, Halil Akin, Mandeep Baines, Louis Martin, Xing Zhou, Punit Singh Koura, Brian O'Horo, Jeff Wang, Luke Zettlemoyer, Mona T. Diab, Zornitsa Kozareva, and Ves Stoyanov. Efficient large scale language modeling with mixtures of experts. *CoRR*, abs/2112.10684, 2021. URL https://arxiv.org/abs/2112.10684.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International Conference on Machine Learning*, pp. 2206–2240. PMLR, 2022.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Zewen Chi, Li Dong, Shaohan Huang, Damai Dai, Shuming Ma, Barun Patra, Saksham Singhal, Payal Bajaj, Xia Song, and Furu Wei. On the representation collapse of sparse mixture of experts. *CoRR*, abs/2204.09179, 2022. doi: 10.48550/arXiv.2204.09179. URL https://doi.org/10.48550/arXiv.2204.09179.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *CoRR*, abs/2204.02311, 2022. doi: 10.48550/arXiv.2204.02311. URL https://doi.org/10.48550/arXiv.2204.02311.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1423. URL https://doi.org/10.18653/v1/n19-1423.

Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathy Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V. Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. Glam: Efficient scaling of language models with mixture-of-experts. *CoRR*, abs/2112.06905, 2021. URL https://arxiv.org/abs/2112.06905.

David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. In Yoshua Bengio and Yann LeCun (eds.), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.4314.

William Fedus, Barret Zoph, and Noam M. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *ArXiv*, abs/2101.03961, 2021.

William Fedus, Jeff Dean, and Barret Zoph. A review of sparse expert models in deep learning, 2022. URL https://arxiv.org/abs/2209.01667.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pp. 5484–5495. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.446. URL https://doi.org/10.18653/v1/2021.emnlp-main.446.

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. *http://web.archive.org/save/http://skylion007.github.io/OpenWebTextCorpus*, 2019.

Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, 2020. URL https://arxiv.org/abs/1908.10396.

Suchin Gururangan, Mike Lewis, Ari Holtzman, Noah A. Smith, and Luke Zettlemoyer. Demix layers: Disentangling domains for modular language modeling. *CoRR*, abs/2108.05036, 2021. URL https://arxiv.org/abs/2108.05036.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. *CoRR*, abs/2203.15556, 2022. doi: 10.48550/arXiv.2203.15556. URL https://doi.org/10.48550/arXiv.2203.15556.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.

Sebastian Jaszczur, Aakanksha Chowdhery, Afroz Mohiuddin, Lukasz Kaiser, Wojciech Gajewski, Henryk Michalewski, and Jonni Kanerva. Sparse is enough in scaling transformers. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 9895–9907, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/51f15efdd170e6043fa02a74882f0470-Abstract.html.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Trans. Big Data*, 7(3):535–547, 2021. doi: 10.1109/TBDATA.2019.2921572. URL https://doi.org/10.1109/TBDATA.2019.2921572.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=rkgNKkHtvB.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 8546–8557, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/9d8df73a3cfbf3c5b47bc9b50f214aff-Abstract.html.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam M. Shazeer, and Z. Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *ArXiv*, abs/2006.16668, 2021.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. BASE layers: Simplifying training of large, sparse models. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 6265–6274. PMLR, 2021. URL http://proceedings.mlr.press/v139/lewis21a.html.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. 2019. URL http://arxiv.org/abs/1907.11692.

Sebastian Nagel. Cc-news. *http://web.archive.org/save/http://commoncrawl.org/2016/10/news-dataset-available*, 2016.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on GPU clusters using megatron-lm. In Bronis R. de Supinski, Mary W. Hall, and Todd Gamblin (eds.), *SC '21: The International Conference for High Performance Computing, Networking, Storage and Analysis, St. Louis, Missouri, USA, November 14 - 19, 2021*, pp. 58:1–58:15. ACM, 2021. doi: 10.1145/3458817.3476209. URL https://doi.org/10.1145/3458817.3476209.

Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Stephen Roller, Sainbayar Sukhbaatar, Arthur Szlam, and Jason Weston. Hash layers for large sparse models. *CoRR*, abs/2106.04426, 2021. URL https://arxiv.org/abs/2106.04426.

Roy Schwartz, Jesse Dodge, Noah Smith, and Oren Etzioni. Green ai. *Communications of the ACM*, 63:54 – 63, 2020.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Sainbayar Sukhbaatar, Arthur D. Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In *NIPS*, 2015a.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. *Advances in neural information processing systems*, 28, 2015b.

Sainbayar Sukhbaatar, Edouard Grave, Guillaume Lample, Herve Jegou, and Armand Joulin. Augmenting self-attention with persistent memory. *arXiv preprint arXiv:1907.01470*, 2019.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys (CSUR)*, 2020.

Trieu H. Trinh and Quoc V. Le. A simple method for commonsense reasoning. *ArXiv*, abs/1806.02847, 2018.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *ArXiv*, abs/1706.03762, 2017.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *CoRR*, abs/2206.07682, 2022. doi: 10.48550/arXiv.2206.07682. URL https://doi.org/10.48550/arXiv.2206.07682.

Guillaume Wenzek, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzm'an, Armand Joulin, and Edouard Grave. Ccnet: Extracting high quality monolingual datasets from web crawl data. In *LREC*, 2020.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. OPT: open pre-trained transformer language models. *CoRR*, abs/2205.01068, 2022a. doi: 10.48550/arXiv.2205.01068. URL https://doi.org/10.48550/arXiv.2205.01068.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. MoEfication: Transformer feed-forward layers are mixtures of experts. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 877–890, Dublin, Ireland, May 2022b. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.71. URL https://aclanthology.org/2022.findings-acl.71.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Y. Zhao, Andrew M. Dai, Zhifeng Chen, Quoc Le, and James Laudon. Mixture-of-experts with expert choice routing. *CoRR*, abs/2202.09368, 2022. URL https://arxiv.org/abs/2202.09368.

Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 19–27, 2015.

A Training Setting

A.1 Hyperparameters

Table 6 specifies shared hyperparameters across all experiments, in which Table 5a contains ones for training data, optimizer, and efficient infrastructure techniques; and Table 5b for architecture. Then, Table 6a describes the hyperparameters specifically for Switch, Table 6b for LoRKM, Table 6c for PKM,

A.2 Data

Here is a detailed description of our pretraining corpus.

- **BookCorpus** (Zhu et al., 2015) consists of more than 10K unpublished books (4GB);
- **English Wikipedia**, excluding lists, tables and headers (12GB);
- **CC-News** (Nagel, 2016) contains 63 millions English news articles crawled between September 2016 and February 2019 (76GB);
- **OpenWebText** (Gokaslan & Cohen, 2019), an open source recreation of the WebText dataset used to train GPT-2 (38GB);