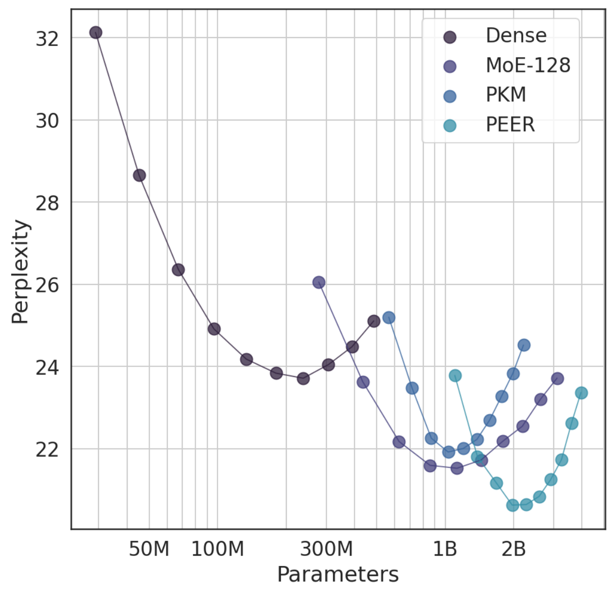
Mixture of A Million Experts

Note: A summary is provided at page 9. It is recommended to go through it before reading the paper in detail.

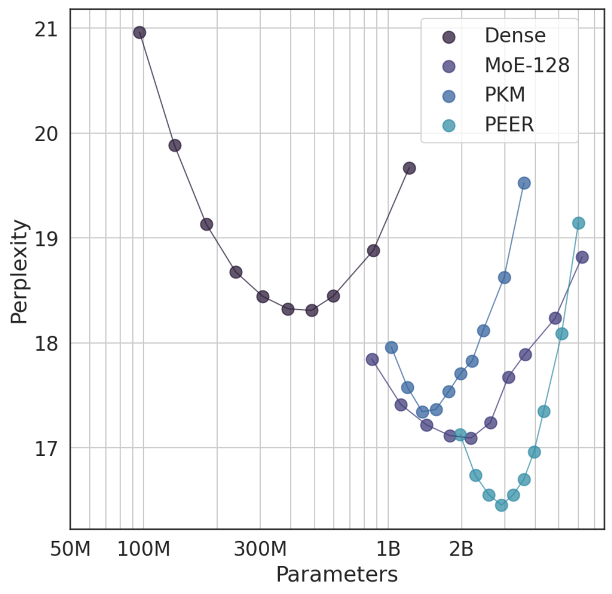
Xu Owen He
Google DeepMind
hexu@google.com

Abstract

The feedforward (FFW) layers in standard transformer architectures incur a linear increase in computational costs and activation memory as the hidden layer width grows. Sparse mixture-of-experts (MoE) architectures have emerged as a viable approach to address this issue by decoupling model size from computational cost. The recent discovery of the fine-grained MoE scaling law shows that higher granularity leads to better performance. However, existing MoE models are limited to a small number of experts due to computational and optimization challenges. This paper introduces PEER (parameter efficient expert retrieval), a novel layer design that utilizes the product key technique for sparse retrieval from a vast pool of tiny experts (over a million). Experiments on language modeling tasks demonstrate that PEER layers outperform dense FFWs and coarse-grained MoEs in terms of performance-compute trade-off. By enabling efficient utilization of a massive number of experts, PEER unlocks the potential for further scaling of transformer models while maintaining computational efficiency.



(a) 6*e*18 FLOPs



(b) 2*e*19 FLOPs

Figure 1: Isoflop comparison on the C4 dataset between PEER and other baselines with two different FLOP budgets (6e18 and 2e19 FLOPs). The x axis is in log scale.

1 Introduction

The past few years have seen the power of scaling (Kaplan et al., 2020; Hoffmann et al., 2022): increasing the number of parameters, amount of training data, or the computational budget has proven to be a reliable way to improve model performance. Notably, feedforward (FFW) layers, responsible for storing factual knowledge (Geva et al., 2021; Dai et al., 2022), account for two-thirds of the total parameters in a transformer. However, one drawback of these dense FFWs is that their computational footprint (FLOPs and device memory consumption) is linearly proportional to their parameter count.

Why Ffn is important but is a mess?

To break the coupling between computational cost and parameter count, many recent works (Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2022; Zhou et al., 2022) have adopted the Mixture-of-Experts (MoE) architecture, which uses a set of sparsely activated expert modules (often FFWs) in place of a single dense FFW. Clark et al. (2022) studied the scaling law of MoE language models and showed that increasing the number of experts is an effective way to improve performance without increasing the inference cost. However, their experiments showed that the efficiency gains provided by MoEs plateau after a certain model size is reached. More recently, Krajewski et al. (2024) discovered that this plateau was caused by using a fixed number of training tokens. When the number of training tokens is compute-optimal, MoEs consistently outperform dense models in terms of FLOP efficiency. Moreover, they introduced granularity (the number of active experts) as a new scaling axis and empirically showed that using higher granularity improves performance. Extrapolating this fine-grained MoE scaling law suggests that continued improvement of model capacity will ultimately lead to a large model with high granularity, corresponding to an architecture of an immense number of tiny experts.

Granularity and scaling laws for MoE

Beyond efficient scaling, another reason to have a vast number of experts is lifelong learning, where MoE has emerged as a promising approach (Aljundi et al., 2017; Chen et al., 2023; Yu et al., 2024; Li et al., 2024). For instance, Chen et al. (2023) showed that, by simply adding new experts and regularizing them properly, MoE models can adapt to continuous data streams. Freezing old experts and updating only new ones prevents catastrophic forgetting and maintains plasticity by design. In lifelong learning settings, the data stream can be indefinitely long or never-ending (Mitchell et al., 2018), necessitating an expanding pool of experts.

Although both efficient scaling and lifelong learning require MoE designs capable of handling a vast number of experts, to the best of our knowledge, the only architecture supporting more than ten thousands of experts is the Mixture of Word Experts (MoWE) (dos Santos et al., 2023). However, MoWE is language-specific and uses a fixed routing scheme. Theoretical and empirical evidence (Clark et al., 2022; Dikkala et al., 2023) highlights the advantages of learned routers over non-trainable ones. Thus, an MoE design with a learned router scalable to over a million experts remains an open area for exploration.

This work introduces the Parameter Efficient Expert Retrieval (PEER) architecture, leveraging product key retrieval (Lample et al., 2019) for efficient routing to an extremely large number of experts, decoupling computational cost from parameter count. This design demonstrates a superior compute-performance trade-off in our experiments, positioning it as a competitive alternative to dense FFW layers for scaling foundation models. The main contributions of this work are:

- **Exploration of Extreme MoE Setting:** Deviating from the focus on a small number of large experts in previous MoE research, this work investigates the under-explored case of numerous tiny experts.
- **Learned Index Structure for Routing:** Demonstrating for the first time that a learned index structure (Kraska et al., 2018) can efficiently route to over a million experts.
- **New Layer Design:** Combining product key routing with single-neuron experts, we introduce the PEER layer that expands layer capacity without significant computational overheads. Empirical results demonstrate its superior efficiency compared to dense FFW, coarse-grained MoEs and Product Key Memory (PKM) layers.
- **Comprehensive Ablation Studies:** We investigate the impact of different design choices of PEER such as number of experts, active parameters, number of heads and query batch normalization on language modeling tasks.

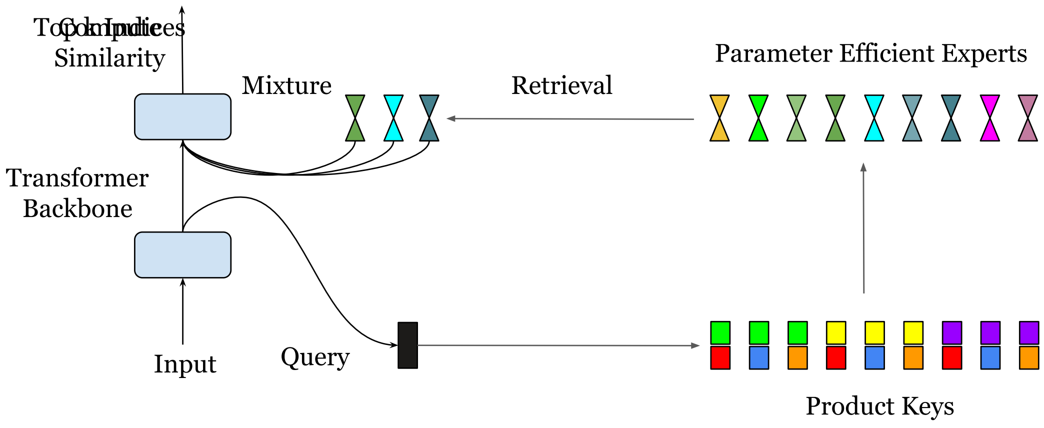


Figure 2: **Illustration of the PEER layer.** A PEER layer can be inserted in the middle of a transformer backbone or can be used to replace FFW layers. Given the state vector x from the previous layer, a query network q maps it to a query vector $q(x)$, which is then compared with the product keys to compute the router scores and to retrieve the top k experts $e_1, ..., e_k$. After the retrieved experts make their predictions $e_i(x)$, their outputs are linearly combined using the softmax-normalized router scores as weights.

2 Method

In this section, we introduce the Parameter Efficient Expert Retrieval (PEER) layer, which is a Mixture of Experts architecture using product keys (Lampe et al. 2019) in the router and single-neuron MLPs as experts. Fig. 2 illustrates the computational process within a PEER layer.

PEER Overview Formally, a PEER layer is a function $f : \mathbb{R}^n \to \mathbb{R}^m$ that consists of three parts: a pool of N experts $\mathbb{E} := \{e_i\}_{i=1}^N$, where each expert $e_i : \mathbb{R}^n \to \mathbb{R}^m$ shares the same signature as f, a corresponding set of N product keys $\mathbb{K} := \{k_i\}_{i=1}^N \subset \mathbb{R}^d$, and a query network $q : \mathbb{R}^n \to \mathbb{R}^d$ that maps the input vector $x \in \mathbb{R}^n$ to a query vector $q(x)$. Let $\mathcal{T}_k$ denote the top-k operator. Given an input x, we first retrieve a subset of k experts whose corresponding product keys have the highest inner products with the query $q(x)$.

$$\boxed{\mathbb{I} = \mathcal{T}_k\left(\{q(x)^T k_i\}_{i=1}^N\right)} \qquad \text{\# Retrieve top } k \text{ experts} \tag{1}$$

Then we apply nonlinear activations (such as softmax or sigmoid) to the query-key inner products of these top k experts to obtain the router scores.

$$\boxed{g_i(x) = s(q(x)^T k_i)} \qquad \text{\# Compute router scores} \tag{2}$$

Finally, we compute the output by linearly combining the expert outputs weighted by the router scores.

$$\boxed{f(x) = \sum_{i \in \mathbb{I}} g_i(x) e_i(x)} \qquad \text{\# Aggregate expert outputs} \tag{3}$$

Product Key Retrieval Since we intend to use a very large number of experts ($N \geq 10^6$), naively computing the top k indices in Eq. 1 can be very expensive. Hence we apply the product key retrieval technique here. Instead of using N independent d-dimensional vectors as our keys k_i, we create them by concatenating vectors from two independent sets of $\frac{d}{2}$-dimensional sub-keys $\mathbb{C}, \mathbb{C}' \subset \mathbb{R}^{\frac{d}{2}}$:

$$\mathbb{K} = \left\{ \begin{bmatrix} c \\ c' \end{bmatrix} | c \in \mathbb{C}, c' \in \mathbb{C}' \right\} \tag{4}$$

Note that here $\mathbb{C}, \mathbb{C}'$ have cardinality $\sqrt{N}$ and c, c' have dimensionality $\frac{d}{2}$. So in practice, we choose N to be a perfect square and d to be an even number.

Efficient top_k retrieval

This Cartesian product structure of $\mathbb{K}$ allows us to find the top k experts efficiently. Instead of comparing $q(x)$ to all N keys in $\mathbb{K}$ and selecting the top k matches, we can split the query vector $q(x)$ into two sub-queries q_1 and q_2 and apply the top k operations to the inner products between the sub-queries and sub-keys respectively:

$$\boxed{\mathbb{I}_{\mathbb{C}} = \mathcal{T}_k\left((q_1^T c_i)\right), \qquad \mathbb{I}_{\mathbb{C}'} = \mathcal{T}_k\left((q_2^T c_j')\right)} \tag{5}$$

This results in a set of k^2 candidate keys $\mathbb{K}' := \{\begin{bmatrix} c_i \\ c_j \end{bmatrix} | i \in \mathbb{I}_{\mathbb{C}}, j \in \mathbb{I}_{\mathbb{C}}'\}$, and it is mathematically guaranteed that the k most similar keys to $q(x)$ from $\mathbb{K}$ are in this candidate set. Moreover, the inner product between the candidate key and $q(x)$ is simply the sum of inner products between the sub-keys and sub-queries: $q(x)^T \begin{bmatrix} c_i \\ c_j \end{bmatrix} = q_1^T c_i + q_2^T c_j$. Hence we can apply the top-k operator again to these k^2 inner products to get the top k matching keys from the original set of product keys $\mathbb{K}$. As explained in Lample et al. (2019). This reduces the complexity of top k expert retrieval in Eq. 1 from $O(Nd)$ as done naively by exhaustive search to $O((\sqrt{N} + k^2)d)$.

Parameter Efficient Experts and Multi-Head Retrieval Unlike other MoE architectures, which often set the hidden layer of each expert to the same size as other FFW layers, in PEER, every expert e_i is a singleton MLP, in other words, it has only one hidden layer with a single neuron:

$$\boxed{e_i(x) := \sigma(u_i^T x) v_i} \tag{6}$$

where v_i, u_i are not matrices but vectors with the same dimension as x, and σ is a nonlinear activation function such as ReLU or GELU. We omit bias terms here for brevity.

Instead of varying the size of individual experts, we adjust the expressiveness of a PEER layer by using multi-head retrieval, similar to the multi-head attention mechanism in transformers and the multi-head memory in PKMs. In particular, we use h independent query networks instead of one, each computes its own query and retrieves a separate set of k experts. However, different heads share the same pool of experts with the same set of product keys. The outputs of these h heads are simply summed up:

$$\boxed{f(x) := \sum_{i=1}^{h} f^i(x) = \sum_{i=1}^{h} \sum_{j \in \mathbb{I}^i} g_j(x) e_j(x)} \tag{7}$$

One can verify that when only one expert is retrieved ($k = 1$) per head, using a PEER layer with h heads is the same as using one expert with h hidden neurons:

$$\boxed{f(x) = \sum_{i=1}^{h} e^i(x) = \sum_{i=1}^{h} \sigma(u_i^T x) v_i = V\sigma(W^T x);} \tag{8}$$

where $W = [u_1, \cdots, u_h], V = [v_1, \cdots, v_h]$. In other words, PEER dynamically assembles an MLP with h neurons by aggregating h singleton MLPs retrieved from a shared repository. Compared to existing MoE approaches that use MLPs with multiple hidden neurons as experts, this design allows shared hidden neurons among experts, enhancing knowledge transfer and parameter efficiency.

Algorithm 1 shows a simplified implementation of the PEER forward pass, storing parameter-efficient expert weights in embedding layers and combining them with einsum operations. This implementation can be easily extended to experts of the GLU variants (Shazeer, 2020) by adding additional linear gating weights. In practice, an efficient implementation may require specialized hardware kernels to accelerate embedding lookup and fusion with the einsum operations.

Why A Large Number of Small Experts? Given an MoE layer, we can characterize it by three hyperparameters: the total number of parameters P, the number of active parameters per token P_{active} and the size of a single expert P_{expert}. Krajewski et al. (2024) showed that the scaling law of MoE models has the following form:

$$\boxed{\mathcal{L}(P, D, G) = c + (\frac{g}{G^\gamma} + a)\frac{1}{P^\alpha} + \frac{b}{D^\beta},} \tag{9}$$

where $\mathcal{L}$ is the final test loss, $a, b, g, \gamma, \alpha, \beta$ are constants, D is the total number of training tokens and the granularity G is the number of active experts:

$$\boxed{G := \frac{P_{\text{active}}}{P_{\text{expert}}}} \tag{10}$$

In order to improve model performance, we need to scale up P, D, G. On the other hand, it is essential to limit P_{active} because the computational and memory costs are primarily determined by the active parameters during training and inference. Notably, the memory footprint corresponding to P_{active} has to be multiplied by the number of tokens in a batch, while the memory cost of P is independent of the batch size and sequence length because only one copy of the model needs to be stored.

Reason why is this an imp metric

As a result, we want to increase P, G but not P_{active}. Since the expert size $P_{\text{expert}} = P_{\text{active}}/G$ and the number of experts $N = P/P_{\text{expert}} = P \cdot G/P_{\text{active}}$, this implies that we should decrease the size of each expert, P_{expert}, and increase the number of experts N. Hence we need a large number of small experts.

In general, for experts that are MLPs with a single hidden layer. $P_{\text{expert}} = (2d_{\text{model}} + 1)d_{\text{expert}}$ and $P_{\text{active}} = (2d_{\text{model}} + 1)d_{\text{active}}$, where d_{model}, d_{expert} and d_{active} are the hidden dimension of the transformer, the number of hidden neurons used in one expert and the total number of hidden neurons activated per token, respectively.

In the case of PEER, we use the smallest expert size possible by setting $d_{\text{expert}} = 1$, and the number of activated neurons is the number of retrieval heads multiplied by the number of experts retrieved per head: $d_{\text{active}} = hk$. Consequently, the granularity of PEER is always $G = P_{\text{active}}/P_{\text{expert}} = d_{\text{active}}/d_{\text{expert}} = hk$.

```
def peer_forward(self, x):
    # Embedding layers storing the down/up projection weights of all experts
    self.w_down_embed = nn.Embed(num_embeddings=self.n_experts, features=self.d_model)
    self.w_up_embed = nn.Embed(num_embeddings=self.n_experts, features=self.d_model)

    # Retrieve the weights of the top matching experts using product keys
    # indices and scores have the shape 'bthk', where h is the number of heads
    indices, scores = self.get_indices(self.query_proj(x), self.sub_keys, top_k=self.k)
    w_down = self.w_down_embed(indices)
    w_up = self.w_up_embed(indices)

    # Compute weighted average of expert outputs
    x = jnp.einsum('btd, bthkd->bthk', x, w_down)
    x = self.activation(x)
    x = x * nn.softmax(scores)
    x = jnp.einsum('bthk, bthkd->btd', x, w_up)
    return x
```

Algorithm 1: Pseudo code implementation of a PEER layer forward pass. An example implementation of the get_indices and query_proj functions in Pytorch can be found in Lample et al. (2021)

3 Experiments

3.1 Pretraining isoFLOP Analysis

We compare PEER with various baselines using isoFLOP analysis (Borgeaud et al., 2022b). We chose a fixed FLOP budget ($6e18$ and $2e19$) and jointly varied the model size and the number of training tokens from the C4 dataset (Raffel et al., 2020) to obtain isoFLOP curves. Each point on an isoFLOP curve has the same computational cost, and we plot them in terms of their model size and final validation perplexity on C4.

For the dense baselines, we varied their size by changing the number of layers, attention heads and model dimensions. For MoE, PKM and PEER methods, we took each of the dense models considered and replaced the FFW layer in the middle block (e.g. in a 12 block transformer, we replace the FFN in block 6) by a layer of MoE, PKM and PEER, respectively.

In MoE, we used the expert-choice (Zhou et al., 2022) routing algorithm, which effectively addresses the expert load imbalance issue and generally outperforms token-choice MoEs (see Section 4 for a review and comparison of these approaches). Each expert has the same size as the original MLPs in the corresponding dense model, and we use 128 experts to cover the same range of model sizes as our PEER models. This type of MoE represents standard coarse-grained MoE approaches, which consist of a small number of large experts.

In PKM, we used 1024^2 memories with $h = 8$ heads and top $k = 32$ memories were selected per head. We also applied query batch normalization, as recommended in the original PKM paper (Lample et al., 2019), to enhance memory usage.

In PEER, we used 1024^2 experts with $h = 8$ heads and top $k = 16$ experts per head. By default, we also enabled query BatchNorm to increase expert usage. Ablation studies in subsection 3.3 investigate the effect of these hyperparameters. Unlike the expert-choice MoE baseline, PEER represents a fine-grained approach where a large number of small experts are employed.

Across all model sizes and methods, we maintained a consistent batch size (128) and sequence length (2048). We calculated the number of training steps by dividing the total compute budget by the FLOPs per training step. Fig. 1 presents the isoFLOP profiles. Compared to the dense FFW baseline, the sparse alternatives shift the isoFLOP curves downward and to right because they introduce a larger number of total parameters P but utilize a smaller or equal number of active parameters P_{active}. Given the same compute budget, a PEER model achieves the lowest compute-optimal perplexity.

3.2 Evaluation on Language Modeling Datasets

After determining the compute-optimal model for each method based on the isoFLOP curves, we evaluated the performance of these pretrained models on several popular language modeling datasets, including Curation Corpus (Curation, 2020), Lambada (Paperno et al., 2016), the Pile (Gao et al., 2020), Wikitext (Merity et al., 2016) and the pretraining dataset C4. Table 1 presents a summary of the evaluation results. We grouped the models based on their FLOP budgets used during training.

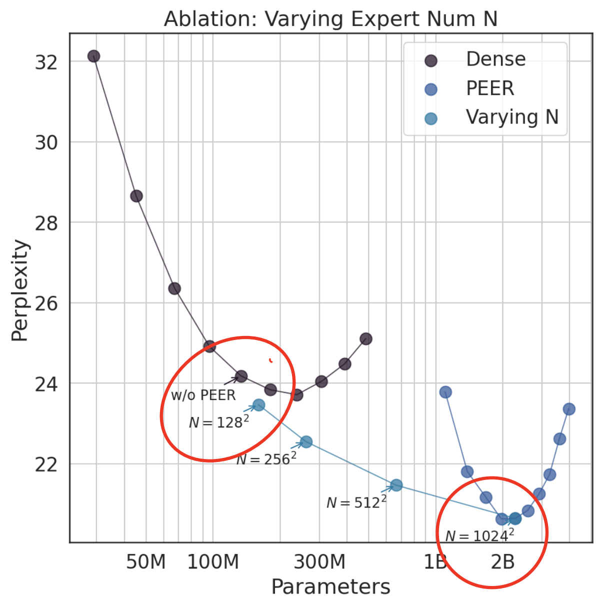
Table 1: Perplexities of the compute-optimal models of each method on language modeling datasets.

Method	Curation Corpus	Lambada	Pile	Wikitext	C4
Dense (6e18)	23.26	21.95	24.55	29.14	23.84
MoE (6e18)	20.98	19.09	23.26	26.10	21.41
PKM (6e18)	21.80	19.39	20.49	27.09	21.92
PEER (6e18)	**20.68**	**17.65**	**19.01**	**25.48**	**20.63**
Dense (2e19)	17.70	12.28	18.19	21.21	18.31
MoE (2e19)	16.88	12.97	17.41	20.28	17.12
PKM (2e19)	17.03	11.18	16.34	20.26	17.36
PEER (2e19)	**16.34**	**10.33**	**14.99**	**19.09**	**16.45**

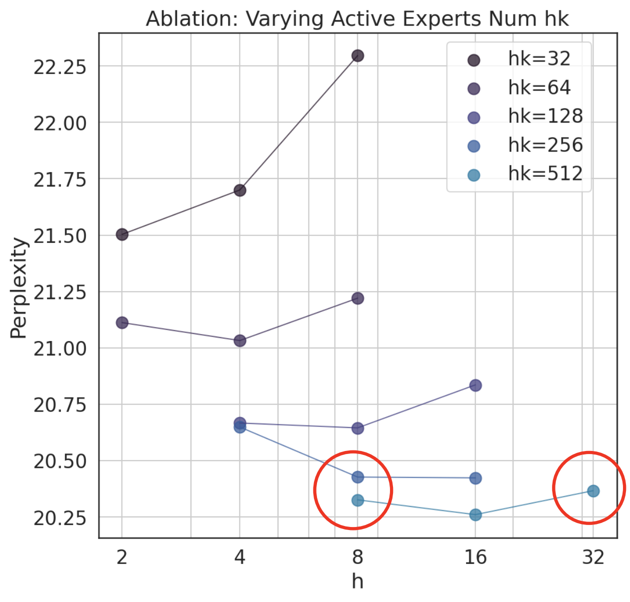
3.3 Ablations

Varying the Number of Total Experts The models in the isoFLOP plot depicted in Fig. 1 all have over a million (1024^2) experts. Here we conduct an ablation study on the effect of the number of experts N, which determines the total parameter count P in Eq. 9. We selected the model at the isoFLOP-optimal position and vary the number of experts ($N = 128^2, 256^2, 512^2, 1024^2$) in the PEER layer while keeping the number of active experts constant ($h = 8, k = 16$). The results are shown in Fig. 3 (a). As can be seen, the isoFLOP curve interpolates between the PEER model with 1024^2 experts and the corresponding dense backbone without replacing the FFW layer in the middle block by a PEER layer. This demonstrates that simply increasing the number experts can improve model performance.

Varying the Number of Active Experts We also conducted an ablation study on the effect of the number of active experts hk, which equals the granularity G in Eq. 9. We systematically varied the number of



(a) Varying Total Expert Num



(b) Varying Active Expert Num

Figure 3: We conduct two ablation studies using the same PEER model configuration. In (a), we vary the total number of experts N while keeping the same number of active experts $hk = 128$. In (b), we vary the number of active experts $G = hk$ by jointly changing h and k while keeping the total number of experts at $N = 1024^2$.

active experts ($hk = 32, 64, 128, 256, 512$) while keeping the number of total experts constant ($N = 1024^2$). Furthermore, for a given hk, we jointly varied h and k to identify the optimal composition. The resulting isoFLOP curves, plotted over the number of heads (h), are shown in Fig. 3 (b).

The results indicate that, within the range of values considered, higher hk generally leads to improved performance. Notably, the optimal h increases as hk increases. However, the performance gradually saturates, and increasing the number of active experts also increases device memory consumption and may necessitate additional accelerator devices. Thus in practice, the appropriate hk values should be selected based on the trade-off between performance, device number and computational resource requirements.

Table 2: **KL and expert usage for different memory sizes, with and without query BN.** Similar to the findings in PKM, using query BN results in a more balanced usage of the experts.

Expert num N	16k		65k		262k		1M	
BatchNorm	No	Yes	No	Yes	No	Yes	No	Yes
Perplexity	23.47	23.47	22.61	22.55	21.54	21.47	20.73	20.64
Expert Usage (%)	100.0	100.0	100.0	100.0	100.0	100.0	99.8	100.0
Unevenness (↓)	0.45	0.30	0.63	0.44	0.97	0.66	1.52	1.06

Expert Usage and Query Batch Normalization Given the presence of over a million experts in the PEER layer, it is natural to inquire how many of these experts are actually selected during inference and whether their usage is evenly distributed. To analyze this, we kept an accumulated router score, denoted as $z'_i = \sum_x g_i(x)$ for each expert e_i across all tokens x within the C4 validation set. Here $g_i(x)$ is the router score used to aggregate the expert output when token x is given as input, with $g_i(x) = 0$ if expert e_i is not selected. From these accumulated router scores, we can obtain an empirical probability distribution vector, denoted as $z = z'/||z'||_1$, representing the distribution of all experts over the C4 validation set. Then we computed the following metrics proposed by Lample et al. (2019) to assess the usage and distribution of experts:

- *Expert Usage*: the fraction of experts retrieved during inference: $\#\{z_i \neq 0\}$
- *Unevenness*: KL divergence between z and the uniform distribution: $\log(N) + \sum_i z_i \log(z_i)$

where N is the number of total experts.

By default, we also added a batch normalization (BN) layer on top of the query network, as proposed by Lample et al. (2019) to increase the expert usage during training. Here we study the effect of adding this BN layer on the above-mentioned metrics.

Table 2 presents the expert usage and unevenness for varying numbers of experts, with and without BN. We can see that even for 1M experts, the expert usage is close to 100%, and using BN can lead to more balanced utilization of the experts and lower perplexities. These findings demonstrate the effectiveness of the PEER model in utilizing a large number of experts.

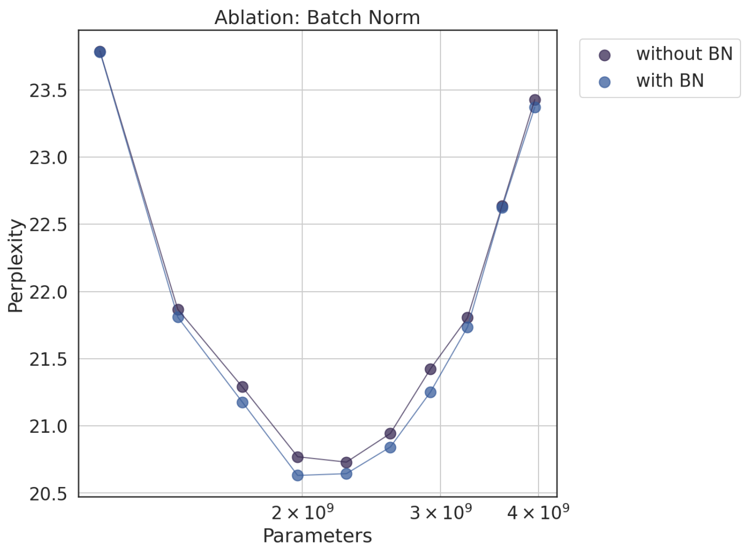


Figure 4: **Query BatchNorm Ablation.** IsoFLOP curves of a PEER model with 1M experts on the C4 dataset, with and without query BatchNorm.

We additionally compared isoFLOP curves with and without BN. Fig. 4 shows that the PEER model with BN generally achieves lower perplexities. While the difference is not significant, it is most pronounced around the isoFLOP-optimal region.

4 Related Works

Mixture of Expert Since Shazeer et al. (2017) demonstrated the effectiveness of sparsely-gated Mixtures of Experts (MoEs) in efficiently increasing model capacity on GPU clusters, MoEs have emerged as a popular technique for scaling large models efficiently. Subsequent research (Fedus et al., 2022; Lepikhin et al., 2020; Du et al., 2022) has proposed variations to address challenges such as load balancing, communication overhead, and training instability. These methods usually replace feedforward (FFW) layers in certain Transformer blocks with sparsely-gated MoE layers, which consist of multiple FFW layers as experts. Typically each expert matches the size of the regular dense FFW layer. Gating scores are calculated for each expert and token, and only the top k experts are activated for each token. These methods are known as token-choice methods. More recently, Zhou et al. (2022) introduced the Expert Choice routing method, where experts choose the top k tokens instead of tokens selecting experts. However, both token-choice and expert-choice methods require the top-k operator on a gating score matrix of size $N \times M$ (N: number of experts, M: number of tokens), resulting in a routing cost of at least $O(N)$. This limits their practical application to a small number of experts (typically less than 128).

Instead of using the top-k operator, some works also proposed using deterministic hash tables as routers (Roller et al., 2021; dos Santos et al., 2023). With $O(1)$ average lookup complexity, these methods offer potential scalability to a large number of experts. However, these routers are fixed and not learned. Clark et al. (2022) showed that deterministic routing does not scale as well as trainable routers. Furthermore, Dikkala et al. (2023) proved theoretically that learned routers offer non-trivial advantages over their fixed counterparts, such as removing spurious directions and identifying latent clusters in data. In contrast to previous works, the proposed PEER layer employs a learned router with sublinear ($O(\sqrt{N})$) complexity.

Since PEER uses lightweight experts, our work is also related to recent studies on parameter-efficient MoEs (Wang et al., 2022; Zadouri et al., 2024). These methods utilize parameter efficient fine-tuning (PEFT) adapters as experts instead of full-sized FFWs. Their focus is on minimizing the number of parameters updated during fine-tuning, allowing storage of only one copy of the large backbone model. In PEER, parameter efficiency refers to the small number of active parameters in the MoE layer, which directly affects FLOPs and activation memory consumption during pre-training and inference. However, PEER could potentially be adapted to retrieve a large number of PEFT adapters.

Retrieval-Augmented Models Our proposed method, with its retrieval mechanism for a large number of experts, aligns with the emerging field of retrieval-augmented models. These models facilitate large model memorization by retrieving knowledge from external databases, leading to improved accuracy and efficiency on knowledge-intensive tasks. Some notable works in this domain include ones by Khandelwal et al. (2019); Borgeaud et al. (2022a); Guu et al. (2020). While these methods retrieve data in various formats, for instance, tokens (Khandelwal et al., 2019), chunks (Borgeaud et al., 2022b) or knowledge graphs (Kang et al., 2023) (see (Gao et al., 2023) for a comprehensive survey on this topic), they differ from the proposed method in that they retrieve data rather than learned functions (experts). This distinction sets our parameter-efficient expert retrieval approach apart from existing retrieval-augmented models.

Efficient Feedforward Layers Enhancing the efficiency of feedforward networks has been a long-standing area of research. Similar to PEER, most approaches are based on the idea of conditional computation (Bengio, 2013), where a gating mechanism is trained to determine which subset of neurons to compute. For instance, Davis & Arel (2013) utilized low-rank weight matrix approximation to estimate the sign of pre-nonlinearity activations. Neurons with negative activations are omitted as they will produce zeros after the nonlinearity. Bengio et al. (2015) explored reinforcement learning to develop an activation-dependant policy for dropping blocks of neurons. More recently, Belcak & Wattenhofer (2023) introduced the Fast FeedForward (FFF) layer that employs a differentiable balanced binary tree to select a neuron block for computation. During inference, only one leaf (corresponding to one block) is selected, hence it has $O(\log(N))$ complexity, where N is the total number of blocks in the tree. However, during training, all leaves and intermediate nodes are activated for gradient calculation, imposing a training complexity of $O(N)$ and limiting the total number of blocks. The most relevant work to ours is the Product Key Memory (PKM) (Lample et al., 2019), whose retrieval technique is utilized as the router in the PEER layer. However, PKM retrieves memory vectors instead of functions, thus their values cannot vary according to the inputs. As we show in Section 3, by changing the memory vectors to input-dependent expert networks, PEER can achieve significantly higher efficiency than PKM. Finally, Csordás et al. (2023) presented a unified view encompassing FFW, MoE and PKM and proposed to change the router normalization function in MoE and PKM from softmax to sigmoid or ReLU.

5 Conclusion

This work introduces a fine-grained MoE architecture that decomposes an extremely wide dense feedforward layer into a large number of small experts. This design is supported by the recent discovery of the fine-grained MoE scaling law. To overcome the computational overhead of routing to a large number of experts, we apply the product keys to efficiently select a small subset of hidden neurons within a wide MLP layer. Empirical analysis using language modeling tasks demonstrate that given the same compute budget, PEER significantly outperforms dense transformers, coarse-grained MoEs and product key memory layers.

Summary

Introduction

- Scaling the number of parameters in a model, the amount of training data, or the computational budget has proven to be an effective yet simple technique for improving model performance.
- The feedforward (FFW) layers in standard transformer architectures, responsible for storing "factual knowledge", incur a linear increase in computational costs and activation memory as the hidden layer width grows.
- Mixture of Experts (MoE) has become a good alternative for decoupling the computational cost and the parameter count.
- Recent works have found that using more experts (high granularity) increases performance. When the number of training tokens is compute-optimal, MoEs consistently outperform dense models in terms of FLOP efficiency.
- Another reason to have many experts is lifelong learning. It has been proved that by simply adding new experts and regularizing them properly, MoE models can adapt to continuous data streams.

Proposed Method

Introduces the Parameter Efficient Expert Retrieval (PEER) layer, which is a MoE architecture using product keys in the router and single-neuron MLPs as experts. A PEER layer consists of three components: a pool of N experts E, a corresponding set of N product keys K, and a query network q. Here is how it works:

- Given a query x, retrieve a subset of k experts whose corresponding product keys have the highest inner products with the query q(x).
- Apply an activation function like sigmoid or softmax to obtain the router scores for these retrieved top k experts.
- Compute the output by linearly combining the expert outputs weighted by the router scores.

Product-Key Retrieval

- If you have a large number of experts, each with a dimensionality d, then computing the top k indices can be very expensive (O(Nd), N being >= 1e6 in this case).
- The authors apply the product key retrieval technique used in the "Large Memory Layers with Product Keys" paper. Instead of using N independent d-dimensional experts as keys, use two independent subsets, each with dimensionality d/2, to form the keys.

- Similarly, split the query vector q(x) into two sub-queries and then apply the top-k operation to the inner products between the sub-queries and sub-keys.
- This Cartesian product structure of K allows us to find the top k experts efficiently, reducing the complexity from O(Nd) to O((√N + k^2) * d)

If you are thinking why this makes sense, consider doing napkin math by taking an example. Say, N=10000(which are your keys/experts K), d = 512 (dimensionality of query vector, and each key), and you are searching for the top 10 values from the inner product of the query, and the keys.

$$e_i(x) := \sigma(u_i^T x)v_i \tag{6}$$

$$f(x) := \sum_{i=1}^{h} f^i(x) = \sum_{i=1}^{h} \sum_{j\in\mathbb{I}^i} g_j(x)e_j(x) \tag{7}$$

$$f(x) = \sum_{i=1}^{h} e^i(x) = \sum_{i=1}^{h} \sigma(u_i^T x)v_i = V\sigma(W^T x); \tag{8}$$

$$\mathcal{L}(P, D, G) = c + (\frac{g}{G^\gamma} + a)\frac{1}{P^\alpha} + \frac{b}{D^\beta}, \tag{9}$$

Parameter Efficient Experts and Multi-Head Retrieval

- In a PEER layer, every expert e is a hidden layer with a single neuron as shown below in equation 6.
- This is different from the existing MoEs as the hidden layer is often set to the original size of the FFN layer.
- To adjust the reduced expressiveness in the PEER layer, the authors use multi-head retrieval, similar to the multi-head attention mechanism in transformers and the multi-head memory in PKMs.
- h independent query networks instead of one, each computes its query and retrieves a separate set of k experts. However, different heads share the same pool of experts with the same set of product keys. The outputs of these h heads are simply summed up. If one expert is retrieved (k = 1) per head, using a PEER layer with h heads is the same as using one expert with h hidden neurons.

The million-dollar question: Why a large number of small experts?

Look at the scaling law for MoE in equation 9 in the above figure where L is the final test loss, a, b, g, ɣ, α, β are constants, D is the total number of training tokens, and the granularity G is the number of active experts.

G = P (active) / P (expert) where:
P = total number of parameters
P(active) = number of active parameters per token
P (expert) = size of single expert

To improve model performance, we need to scale up P, D, and G. On the other hand, we need to limit P (active) because the computational and memory costs are primarily determined by the active parameters during training and inference. Now,

N = P / P (expert) = P * G / P (active)

This implies that we should decrease the size of each expert, and increase the number of experts N. Also:

P(expert) = (2*d_model + 1) * d_expert
P(active) = (2*d_model +1) * d_active; where
d_model = hidden dimension of the transformer
d_expert = number of hidden neurons used in one expert
d_active = total number of hidden neurons activated per token

For PEER, d_expert is 1, hence the granularity of PEER is always hk where h is the number of heads, and k is the number of experts retrieved per head

Acknowledgments

The author would like to thank Adam Santoro, Arthur Guez, Arthur Szlam, Andrei Rusu, Marc'aurelio Ranzato, Simon Schug, Utku Evci, Doina Precup and Razvan Pascanu for their insightful discussions and invaluable advice. The author is also grateful to Zhitao Gong, Daniel Toyama, Qixuan Feng and Jiajun Shen for their technical assistance. Special thanks are due to Adam Santoro for sharing the isoFLOP analysis scripts and to Andy Brock for building and maintaining the internal codebase used to train the models.

References

Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3366–3375, 2017.

Peter Belcak and Roger Wattenhofer. Fast feedforward networks. *arXiv preprint arXiv:2308.14711*, 2023.

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

Yoshua Bengio. Deep learning of representations: Looking forward. In *International conference on statistical language and speech processing*, pp. 1–37. Springer, 2013.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego De Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 2206–2240. PMLR, 17–23 Jul 2022a. URL https://proceedings.mlr.press/v162/borgeaud22a.html.

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, 2022b.

Wuyang Chen, Yanqi Zhou, Nan Du, Yanping Huang, James Laudon, Zhifeng Chen, and Claire Cui. Lifelong language pretraining with distribution-specialized experts. In *International Conference on Machine Learning*, pp. 5383–5395. PMLR, 2023.

Aidan Clark, Diego De Las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, et al. Unified scaling laws for routed language models. In *International Conference on Machine Learning*, pp. 4057–4086. PMLR, 2022.

Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. Approximating two-layer feedforward networks for efficient transformers. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Curation. Curation corpus base, 2020.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8493–8502, 2022.

Andrew Davis and Itamar Arel. Low-rank approximations for conditional feedforward computation in deep neural networks. *arXiv preprint arXiv:1312.4461*, 2013.

Nishanth Dikkala, Nikhil Ghosh, Raghu Meka, Rina Panigrahy, Nikhil Vyas, and Xin Wang. On the benefits of learning to route in mixture-of-experts models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL `https://openreview.net/forum?id=QV79qiKAjD`.

Cicero Nogueira dos Santos, James Lee-Thorp, Isaac Noble, Chung-Ching Chang, and David Uthus. Memory augmented language models through mixture of word experts, 2023.

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 P Bosma, Zongwei Zhou, Tao Wang, Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. GLaM: Efficient scaling of language models with mixture-of-experts. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 5547–5569. PMLR, 17–23 Jul 2022. URL `https://proceedings.mlr.press/v162/du22c.html`.

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

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.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

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*, pp. 5484–5495, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.446. URL `https://aclanthology.org/2021.emnlp-main.446`.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *International conference on machine learning*, pp. 3929–3938. PMLR, 2020.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Minki Kang, Jin Myung Kwak, Jinheon Baek, and Sung Ju Hwang. Knowledge graph-augmented language models for knowledge-grounded dialogue generation. *arXiv preprint arXiv:2305.18846*, 2023.

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. *arXiv preprint arXiv:2001.08361*, 2020.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations*, 2019.

Jakub Krajewski, Jan Ludziejewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, et al. Scaling laws for fine-grained mixture of experts. *arXiv preprint arXiv:2402.07871*, 2024.

Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 international conference on management of data*, pp. 489–504, 2018.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. *Advances in Neural Information Processing Systems*, 32, 2019.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Minimalist implementation of a product-key memory layer. `https://github.com/facebookresearch/XLM/blob/main/PKM-layer.ipynb`, 2021.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Hongbo Li, Sen Lin, Lingjie Duan, Yingbin Liang, and Ness B. Shroff. Theory on mixture-of-experts in continual learning, 2024. URL `https://arxiv.org/abs/2406.16437`.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Tom Mitchell, William Cohen, Estevam Hruschka, Partha Talukdar, Bishan Yang, Justin Betteridge, Andrew Carlson, Bhavana Dalvi, Matt Gardner, Bryan Kisiel, et al. Never-ending learning. *Communications of the ACM*, 61(5):103–115, 2018.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL `https://www.aclweb.org/anthology/P16-1144`.

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. *Journal of machine learning research*, 21(140):1–67, 2020.

Stephen Roller, Sainbayar Sukhbaatar, Jason Weston, et al. Hash layers for large sparse models. *Advances in Neural Information Processing Systems*, 34:17555–17566, 2021.

Noam Shazeer. Glu variants improve transformer, 2020. URL `https://arxiv.org/abs/2002.05202`.

Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=B1ckMDqlg`.

Yaqing Wang, Sahaj Agarwal, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. AdaMix: Mixture-of-adaptations for parameter-efficient model tuning. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 5744–5760, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.388. URL `https://aclanthology.org/2022.emnlp-main.388`.

Jiazuo Yu, Yunzhi Zhuge, Lu Zhang, Ping Hu, Dong Wang, Huchuan Lu, and You He. Boosting continual learning of vision-language models via mixture-of-experts adapters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 23219–23230, June 2024.

Ted Zadouri, Ahmet Üstün, Arash Ahmadian, Beyza Ermis, Acyr Locatelli, and Sara Hooker. Pushing mixture of experts to the limit: Extremely parameter efficient moe for instruction tuning. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=EvDeiLv7qc`.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.