REM: SPARSIFY AND MOEFY MODELS WITH POST-HOC RELU MODULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

The expanding scale of foundational models poses challenges in computational resources during model training and inference. A promising solution is to exploit contextual sparsity to convert monolithic modules into selectively computed ones like Mixture of Experts (MoE). However, existing conversion methods rely on predicting the activation sparsity in the original modules, which is only applicable to MLP modules and suffers from redundancy or performance degradation caused by inaccurate predictions. Moreover, models with non-ReLU activations either need to undergo a costly ReLUfication process or have lower activation sparsity. We propose that, instead of inducing sparsity in the original module and training the router to predict it, sparsity can be directly created by the router, which does not rely on specific properties of the main module and can have arbitrary granularity. We introduce ReM (ReLU Modulation), which involves training a modulator gated by ReLU that scales the hidden states (or outputs) of the original module to sparsify it. To obtain structured sparsity that enables parallelization, the weights of this modulator can be clustered to convert it into a MoE router. On BERT-base, ReM reduced inference FLOPs by up to 93%—substantially improving upon prior methods—while maintaining comparable accuracy, and achieved these gains with over 99% less retraining costs than previous methods.

1 INTRODUCTION

In recent years, there has been a growing trend toward training larger and more powerful deep learning models. In particular, transformers (Vaswani et al., 2017) have emerged as the architecture of choice in fields such as natural language processing and computer vision. While these models deliver tremendous capabilities, it is becoming increasingly important to reduce the computational cost and memory consumption of their training and inference. In this paper, we focus on the inference computation cost, which can be quantified by the number of FLOPs (or MACs) and the wall-time latency.

One direction to address the challenge is to exploit contextual sparsity—that is, only a subset of the model's components is required for processing a given input (Liu et al., 2023). Recent works have shown some initial success in converting MLP modules into Mixture of Experts (MoE) modules, a process known as MoEfication (Zhang et al., 2022; Szatkowski et al., 2024; Lee et al., 2024). However, existing MoEfiction methods rely on the activation sparsity in the original modules, which means zero or small values induced by the activation functions, because they train the MoE router to predict the activation levels to minimize the deviation from the original model. This results in three limitations: 1. there is a difference between the predicted activation and the true activation, meaning that not all activation sparsity can be effectively utilized; 2. these methods can only be applied to MLPs; 3. models with non-ReLU activations either need to undergo a costly ReLUfication process or have lower activation sparsity.

Inspired by the sparsity induced by the ReLU activation, we propose ReM (**R**eLU **M**odulation), a method that achieves selective computation on inherently dense modules by integrating sparsification and clustering into router training. ReM involves training a small modulator whose outputs, gated by ReLU, scale the main module's output. To achieve structured sparsity for better parallelization, the weights of this modulator can be clustered to form an MoE router, and the cluster labels are used to permute the main module's weights to create a set of experts. This approach eliminates

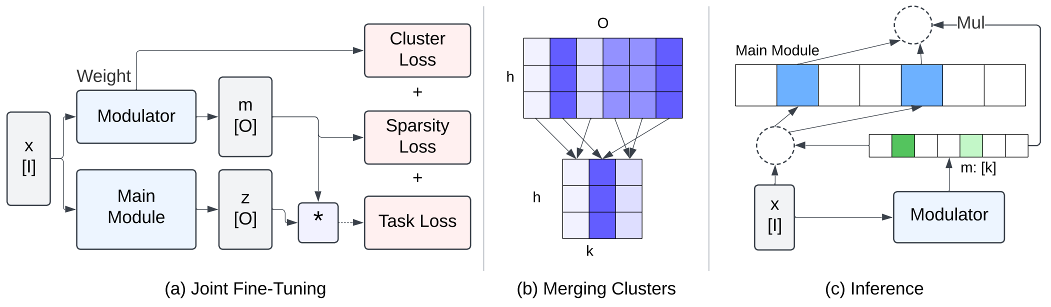

Figure 1: Overview of ReM: (a) we train a small modulator with three joint objectives, (b) then merge the clustered weights in the output layer of the modulator (from $\boldsymbol{W} \in \mathbb{R}^{h \times O}$ to $\boldsymbol{C} \in \mathbb{R}^{h \times k}$. where h is the hidden size of the modulator, $\boldsymbol{W}$ is the original weight matrix, $\boldsymbol{C}$ contains the k cluster centers) to produce a router that outputs a single score for each of the k clusters and permute weights of the main module to produce a MoE, (c) and use the clustered modulator to dynamically decide which parts of the MoEfied main module to compute during inference.

the reliance on existing activation sparsity and the disparity between that and the router's prediction, which allows for higher effective sparsity, application on a wider range of modules (in particular any linear layers), and removal of the process of replacing the activation function.

We evaluated our method on the MLPs and the Query, Key, Value and Output projections in Attention in Bert (Devlin et al., 2019). The resulting model achieved classification accuracy similar to the dense model while reducing the inference FLOPs by 90%, in comparison to 62.6% in D2DMoE (Szatkowski et al., 2024), and achieved these gains with over 99% less retraining costs than previous methods.

2 ReLU Modulation

In this section, we will introduce the core idea of ReLU Modulation, which will then be applied to different modules in the following section. We will commonly refer to input size, hidden size, output size, and the number of clusters as I, H, O, and k respectively. Figure 2 illustrates the difference between ReM and previous works (D2DMoE and MoEfication).

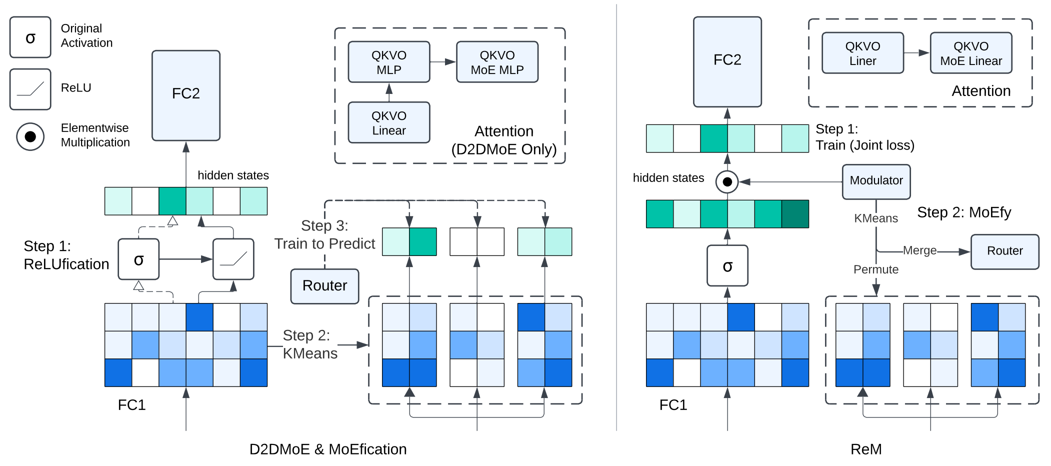

Figure 2: Comparison of ReM with D2DMoE and MoEfication. Only some parts are shown.

2.1 Sparsification

To mitigate performance drop after sparsification and MoEfication, previous works mostly adopted the joint objective of minimizing the deviation of the hidden states from that of the original dense model. We propose to relax this constraint by reframing this objective to produce a sparse model that directly minimizes the task loss, which may produce drastically different hidden states from the dense model. Specifically, in contrast to the previous works that train either a classification or regression model to predict the activation in the original model and only compute neurons with large predicted activation , we propose to remove this supervision and directly modulate the hidden states to minimize jointly the task loss and sparsification loss.

Given a module with a specific output shape, we train a small model, the modulator, whose output is either of the same shape or can be broadcasted to the same shape. The output of the modulator is then passed through a ReLU function to introduce sparsity, and then element-wise multiplied with the output of the module. For a smooth transition, we initialize the modulator in a way that makes the resulting modulation an identity function.

L1 regularization is a common choice to induce sparsity. However, it assigns a uniform gradient to all values. Since the goal of sparsification is to produce as many zeros as possible instead of producing small values, it is better to assign a larger gradient to values closer to zero and reduce the influence on large values. Therefore, we use a L_p regularization loss normalized by the output size: given a modulator output $\boldsymbol{m}$,

$$\mathcal{L}_{\text{sparsity}} = \frac{1}{N}\sum_{i=1}^{N} |\boldsymbol{m}_i|^p, \qquad \frac{\partial \mathcal{L}_{\text{sparsity}}}{\partial \boldsymbol{m}_i} = \begin{cases} \frac{1}{N} p \cdot \boldsymbol{m}_i^{p-1} & \text{if } \boldsymbol{m}_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

where $p \in (0, 1]$ is a hyperparameter, which we set to 0.5; N is the number of neurons. We apply gradient clipping and set the gradient to 0 when $\boldsymbol{m}_i = 0$.

2.2 Clustering

Previous works (Szatkowski et al., 2024; Zhang et al., 2022) employed a separate step of clustering neurons after the sparsification stage using the weight of the first layers (or gate projection in models with GLU activations (Dauphin et al., 2017)) in MLPs. While it was hypothesized that neurons with similar weights would have similar activation patterns, this is suboptimal when the goal is to group together neurons with similar modulations. In addition, as a one-time process, the clustering cannot be changed as the modulators are being trained.

We propose to integrate the clustering step into the modulator training process. Specifically, in each training step, we perform a single iteration of Balanced-KMeans clustering on the weights of output layer in the modulator. Then, we compute a cluster loss defined as the distance between each output weight and its cluster center: $\mathcal{L}_{\text{cluster}} = \frac{1}{N}\sum_{i=1}^{N} ||\boldsymbol{W}_i - \boldsymbol{C}_{l_i}||_1$, where $\boldsymbol{W}_i$ is the weight corresponding to the ith neuron in the output layer of the modulator, and $\boldsymbol{C}_{l_i}$ is the center of its corresponding cluster $\boldsymbol{l}_i$. In late-stage training, the clustering is enforced by cluster mixing, i.e., gradually replacing the modulation of each neuron with the mean modulation in their corresponding cluster. After training, the clusters are merged to convert the modulator into a router, and the cluster labels are used to permute the weights of the main module to obtain an MoE.

2.3 Joint Fine-tuning

Sparsification, clustering, and router training are achieved in a single training cycle: we train the modulator to minimize the joint loss $\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda_{sparsity}\mathcal{L}_{\text{sparsity}} + \lambda_{cluster}\mathcal{L}_{\text{cluster}}$, where L_{task} is the loss on the target task, e.g., language modeling, $\lambda_{sparsity}$ and $\lambda_{sparsity}$ are two hyperparameters. In practice, different schedules are used for the three losses: we start with only the task loss, then warm up the sparsity loss, and finally warm up the cluster loss.

3 Applications

We demonstrate the application of ReM to different modules in this section. Additional designs are shown in the appendix.

3.1 Linear Layers

Due to the reliance on sparsity induced by activation functions, previous works' scope was limited to MLP layers. For instance, D2DMoE had to replace the attention projections with MLPs to convert them into MoE. Without the constraint of predicting sparsity, our method can be applied to arbitrary linear layers. To reduce the size of the modulator, we use a two-layer MLP with a small hidden dimension $h \ll I, O$. As a result, the number of parameters in the modulator is only $h(I+O)$, in contrast to the $I \times O$ parameters in the original linear layer. After clustering, this is further reduced to $h(I+k)$. The original linear layer and the modulated linear layer before the clustering process the input $\boldsymbol{x} \in \mathbb{R}^I$ to produce $\boldsymbol{y} \in \mathbb{R}^O$ as follows:

$$\begin{aligned}&\text{Original:} && \boldsymbol{y} = \boldsymbol{x}\boldsymbol{W} \qquad (\boldsymbol{W} \in \mathbb{R}^{I\times O})\\ &\text{Modulated (Before Clustering):} && \boldsymbol{m} = \text{ReLU}(\text{Act}(\boldsymbol{x}\boldsymbol{W}_d')\boldsymbol{W}_u'), \quad \boldsymbol{y} = \boldsymbol{m} \odot \boldsymbol{x}\boldsymbol{W}\end{aligned}$$

where Act is an arbitrary activation function which we set to SiLU (Elfwing et al., 2018), $\boldsymbol{W}_d' \in \mathbb{R}^{I\times h}$ and $\boldsymbol{W}_u' \in \mathbb{R}^{h\times O}$ are the weight of the two layers in the modulator, $\boldsymbol{m} \in \mathbb{R}^O$ is the neuron-wise modulation, $\odot$ denotes element-wise multiplication. For clarity, we leave out the bias terms, but they are treated in the same way.

After training, we obtain a cluster center matrix $\boldsymbol{C} \in \mathbb{R}^{i\times k}$, where $\boldsymbol{C}_{i,:}$ is the center of the ith cluster, and a cluster label vector $\boldsymbol{l} \in \mathbb{N}^o$, which $\boldsymbol{l}_j$ indicates the label (i.e., cluster index) of the jth neuron. Thus, we can directly replace $\boldsymbol{W}_u'$ with $\boldsymbol{C}$ to obtain a MoE router. We permute the weights of the original linear layer according to the cluster labels to obtain the weights of each expert $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_k$ in the same way as Zhang et al. (2022). Finally, the MoEfied ReM Linear layer is:

$$\boldsymbol{m} = \text{ReLU}(\text{Act}(\boldsymbol{x}\boldsymbol{W}_u')\boldsymbol{C}), \quad \boldsymbol{y}_i = \begin{cases}\boldsymbol{m}_i \odot \boldsymbol{x}\boldsymbol{W}_i & \text{if } \boldsymbol{m}_i > 0\\ 0_{O/k} & \text{otherwise}\end{cases}, \quad \boldsymbol{y} = \text{concat}(\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_k)$$

3.2 MLP

The implementation of ReM on a 2-layer MLP is similar to the linear layer. The modulation is applied on the hidden states, i.e., the output of the first linear layer (or the product of the down projection and gate projection in models with GLU activations). After that, the second linear layer can also be sparsely computed along the input dimension.

We note that it is also possible to apply ReM on the output of the second layer, resulting in sparse computation of this layer along both the input and output dimension, but without efficient implementation and a sufficiently large model, the overhead caused by indexing and launching CUDA kernels may outweigh the additional benefit of sparsity.

3.3 Attention

We propose two variants of ReM for the Attention module: projection modulation and attention score modulation. While we consider the latter to be more promising, it is not directly comparable to our current baselines, so we leave it in Appendix B and intend to add other baselines in the future. In the main text, we focus on the former.

In projection modulation, we convert the Query, Key, Value, and Output (abbreviated as QKVO) projections in the attention module into MoE modules. Each of these projections is a linear layer, so we can directly apply the method introduced in section 3.1, with an optional simplification: since the Q, K, and V projections are already grouped by heads, we can directly set each head to be a cluster and initialize the modulators' output size to the number of heads, then train without the cluster loss.

4 Experiments

4.1 Setup

Generally, we followed the setup for evaluating Bert (Devlin et al., 2019) in D2DMoE.

Datasets We used the Carer emotion classification dataset (Saravia et al., 2018). Following Szatkowski et al. (2024), we padded all sequences to 128 tokens (we note that this sequence length is longer than all sequences in the dataset), and used the training split as both the training set, and the validation split as the test set.[1]

Model and Training Setup We used the Adam optimizer to fine-tune Bert-base-uncased for 10 epochs with a batch size of 48. Figure 3 shows a comparison of training cost with D2DMoE (Szatkowski et al., 2024). We experimented with two granularity: expert size=24 and 128.

Evaluation We report the MACs and accuracy of our fine-tuned model. We compared our method with D2DMoE and MoEfication, the results of which are taken from Szatkowski et al. (2024). We used the same profiler as Szatkowski et al. (2024), namely the *fvcore* Flop Counter[2] [3]

4.2 Results

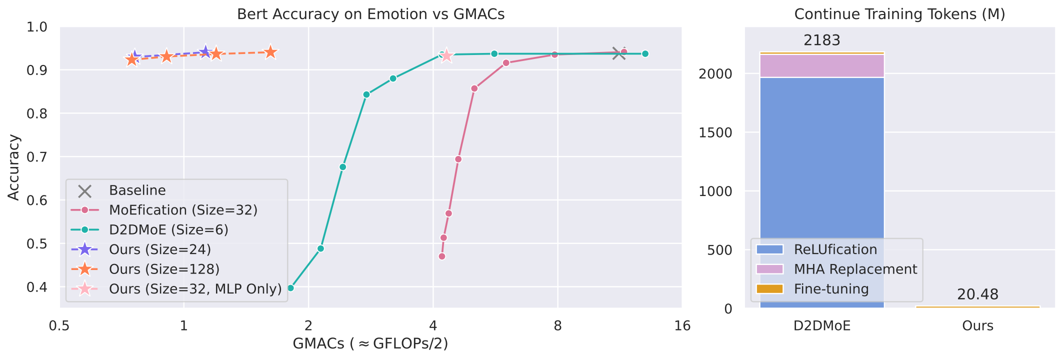

Figure 3: Comparison of ReM with D2DMoE and MoEfication. ReM achieved comparable accuracy with significantly lower FLOPs even with larger expert size (left), while also requiring minimal continue training cost (right). We also report the result of our method without modifying the attention projections (MLP Only).

As shwon in Figure 3, ReM achieved similar accuracy with significantly lower FLOPs compared to the baselines, and can further reduce the FLOPs at the cost of a slight decrease in accuracy (left). In addition, ReM requires minimal retraining cost compared to D2DMoE and MoEfication (right).

4.3 Wall Time Speedup

We evaluated how well the theoretical FLOP reduction translates to wall time speedup. We used the same data as in the main experiments and measured the latency of the models on an A6000 GPU. To evaluate the importance of clustering neurons into experts, we included a setting where MLP and O projection modulators are not clustered. As shown in Table 1, ReM achieved a 2.46× wall time speedup compared to the dense model. The model without clustering (named as ReM (N) in the table) suffered from immense overheads, which caused an 80% slowdown compared to the dense model despite having only 10.6% of the FLOPs, which is also larger than with clustering due to the larger modulator size.

Table 1: Wall time speedup

Model	GMACs	Samples/s
Dense	11.20	735
ReM	0.75	1809
ReM (N)	1.184	140

Although the speedup was evident, it did not fully align with the theoretical gains implied by the MAC reduction. Profiling with the PyTorch Profiler revealed that, after applying ReM, matrix multiplications in MLP and QKVO projections, combined with the modulators' computation, occupied

[1]`https://github.com/bartwojcik/D2DMoE/blob/0d80c763e5/data_utils/data.py#L788-L827`

[2]`https://github.com/facebookresearch/fvcore`

[3]This profiler shows *FLOPs* but actually reports *MACs*, where 1MAC ≈ 2FLOPs.

only about 23.6% of each layer's runtime. In contrast, indexing operations accounted for 18.9%, the remaining attention components comprised 14.8%, and the remaining 42.6% falls into layernorm and miscellaneous overheads beyond the scope of this work (details in Appendix A). We anticipate these overheads to become a smaller fraction of the total runtime in larger models, thus delivering more pronounced speedups in practice.

5 Analysis

We evaluated the effect of the sparsity and cluster loss through ablation studies. The same dataset was used, except that the trivial sparsity caused by the pad tokens was excluded from the results. To better match realistic application scenarios where it is important to parallelize computation, the large expert size setting (24 experts of size 128 in MLP, 12 experts of size 64 in each attention projection) was used.

For our full method, we included two settings: mild ($\lambda_{sparsity} = 1$) and aggressive ($\lambda_{sparsity} = 8$). The results were compared to a model trained without the sparsity loss ($\lambda_{sparsity} = 0$), and models trained without the cluster loss in the aggressive setting. In the latter, we still used the same clustering algorithm and gradual replacement of neuron-level modulation by cluster-level modulation, but set $\lambda_{cluster}$ to 0. For the aggressive setting and the no-cluster-loss setting, we repeated the experiment 4 times with batch sizes 46, 48, 49, and 50, and reported the mean and standard deviation of the accuracy.

Model	**Accuracy**	**MLP%**	**QKVO%**
No Sparsity	94.00	79.73	93.91
ReM (Mild)	94.05	62.65	48.42
ReM (Aggressive)	92.20±0.13	22.36	11.98
No Cluster (Aggressive)	90.57±1.07	16.64	9.79

Table 2: Results of the ablation study. MLP% and QKVO% indicate the percentage of the MLP and QKVO projections being activated (i.e., requires computation).

As shown in Table 2, the sparsity loss significantly increased the sparsity while maintaining the performance in the mild setting. Removing the cluster loss made the training less stable, as depicted by the lower mean and larger standard deviation in accuracy. Intuitively, while the model can learn to adapt to the gradual merger of the clusters, without the clustering loss the weights in each cluster do not become similar to each other, and thus are more likely to be abruptly assigned to different clusters in the next iteration, making the training less stable.

6 Discussion

In this work, we proposed ReM, a method that can achieve sparsification and MoEfication on a modules with no inherent sparsity. We demonstrated the effectiveness of ReM on Bert, achieving similar accuracy with significantly lower FLOPs compared to the baselines, which then led to wall time speedup. We also showed that ReM significantly reduced the retraining cost compared to D2DMoE and MoEfication by removing the dependency on existing sparsity in the original modules.

Our preliminary experiments have centered on Bert, and we have only compared our results to D2DMoE and MoEfication. We plan to perform more extensive experiments on a wider range of models and compare our method to other related works. In particular, we intend to investigate the efficacy of our method on decoder models and non-transformer models and explore the potential of KV-Cache sparsification and sequence compression using the methods introduced in the appendix.

References

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pp. 933–941. PMLR, 2017.

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, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://aclanthology.org/N19-1423/`.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018.

Jaeseong Lee, Seung-won Hwang, Wonpyo Park, and Mingi Ji. Breaking ReLU barrier: Generalized MoEfication for dense pretrained models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 10097–10107, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.563. URL `https://aclanthology.org/2024.emnlp-main.563/`.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pp. 22137–22176. PMLR, 2023.

Artidoro Pagnoni, Ram Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srinivasan Iyer. Byte latent transformer: Patches scale better than tokens, 2024. URL `https://arxiv.org/abs/2412.09871`.

Elvis Saravia, Hsien-Chi Toby Liu, Yen-Hao Huang, Junlin Wu, and Yi-Shin Chen. CARER: Contextualized affect representations for emotion recognition. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3687–3697, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1404. URL `https://www.aclweb.org/anthology/D18-1404`.

Filip Szatkowski, Bartosz Wójcik, Mikołaj Piórczyński, and Simone Scardapane. Exploiting activation sparsity with dense to dynamic-k mixture-of-experts conversion. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

LCM team, Loïc Barrault, Paul-Ambroise Duquenne, Maha Elbayad, Artyom Kozhevnikov, Belen Alastruey, Pierre Andrews, Mariano Coria, Guillaume Couairon, Marta R. Costa-jussà, David Dale, Hady Elsahar, Kevin Heffernan, João Maria Janeiro, Tuan Tran, Christophe Ropers, Eduardo Sánchez, Robin San Roman, Alexandre Mourachko, Safiyyah Saleem, and Holger Schwenk. Large concept models: Language modeling in a sentence representation space, 2024. URL `https://arxiv.org/abs/2412.08821`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Junxiong Wang, Tushaar Gangavarapu, Jing Nathan Yan, and Alexander M. Rush. Mambabyte: Token-free selective state space model, 2024. URL `https://arxiv.org/abs/2401.13660`.

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

A Inference Latency Breakdown

See figure 4.

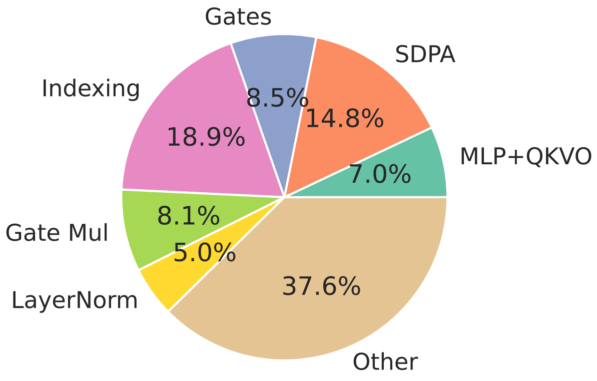

Figure 4: Latency breakdown

B Another Variant of Attention Modulation

In this section, we introduce another variant of ReM for the Attention module, which applies modulation on the attention scores. This variant aligns with the intuition that each attention head only assign meaningful attention scores to a subset of tokens, and the remaining, small attention values can be sparsified. It also has the potential to reduce VRAM consumption by compressing the KV-Cache.

A single Attention head in a classical Multi-Head Attention module (Vaswani et al., 2017) process an input $\boldsymbol{X} \in \mathbb{R}^{l,h}$ by

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

$$\boldsymbol{Z} = \frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}} \quad (\boldsymbol{Z} \in \mathbb{R}^{l,l}) \tag{2}$$

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}(\boldsymbol{L})\boldsymbol{V},$$

$$\text{where softmax}(\boldsymbol{L})_i = \frac{\exp(\boldsymbol{Z}_i)}{\sum \exp(\boldsymbol{Z}_i)} \tag{3}$$

We apply three modulations on Attention: Q, KV, and matrix modulation. For a token and attention head, the Q and KV modulations adjust entire rows and columns of the attention matrix, respectively. Essentially, when the Q modulation is 0 for a Query head, the token cannot attend to any token through that head; when the KV modulation is 0 for a pair of Key-Value heads, the token cannot be attended by any token on that head. Thus, the corresponding computation of QKV projections (equation 1) and KV-cache can be discarded. Notice that the QKV projections are already grouped by heads, so clustering is not needed and we directly initialize the modulators' output dimension to match the number of heads, then train with only the task loss and the sparsity loss. For a model with n_Q Query heads and n_K Key-Value heads, the Q and KV modulations are computed as follows:

$$\boldsymbol{M}_Q \in \mathbb{R}^{l,n_Q} = \boldsymbol{X}\boldsymbol{W}_{Q_M} + \boldsymbol{b}_Q$$

$$\boldsymbol{M}_K \in \mathbb{R}^{l,n_K} = \boldsymbol{X}\boldsymbol{W}_{K_M} + \boldsymbol{b}_K$$

The matrix modulation operates on a more granular level, adjusting the attention matrix element-wise. For clarity, we discuss how it is applied to a single attention head (except for the Q and KV modulators, which directly outputs a score for each head); the same process is repeated for all heads.

The modulator consists of an extra pair of (small) Q and K heads, whose outputs are multiplied to produce attention logits of the same shape as $\boldsymbol{Z}$ by equation 2:

$$\boldsymbol{Q}_M = \boldsymbol{X}\boldsymbol{W}_{Q_M},\ \boldsymbol{K}_M = \boldsymbol{X}\boldsymbol{W}_{K_M}$$

$$\boldsymbol{M}_{mat} = \text{ReLU}(\frac{\boldsymbol{Q}_M \boldsymbol{K}_M^T}{\sqrt{d_k}})$$

This added granularity allows higher sparsity in the attention matrix, and therefore reduces the cost of computing the logits (equation 2). The logits are then passed through a ReLU function instaed of softmax.

The output of the three modulators are merged by broadcasting the Q and KV modulations and then multiplying with the attention matrix:

$$\boldsymbol{M} = \boldsymbol{M}_{mat} \odot (\boldsymbol{M}_Q)_{:,\text{newdim},m} \odot (\boldsymbol{M}_K)_{\text{newdim},:,m}$$

Where m is the index of the current attention head, newdim denotes inserting a new dimension of size 1 and broadcasting. Then, the modulation is applied during the softmax operation. Specifically, we replace the original softmax function (equation 3) with the modulated softmax:

$$\text{MSoftmax}(\boldsymbol{Z}, \boldsymbol{M})_i = \frac{\boldsymbol{M}_i^2 \odot \exp(\boldsymbol{Z}_i)}{\sum \boldsymbol{M}_i \odot \exp(\boldsymbol{Z}_i) + \epsilon}$$

where M is the output of the matrix modulation, ϵ is a small value for avoiding zero division. This function is designed to satisfy the following properties:

- $\text{MSoftmax}(\boldsymbol{Z}, 1_{l\times l}) = \text{softmax}(\boldsymbol{Z})$;
- $\text{MSoftmax}(\boldsymbol{Z}, 0_{l\times l}) = 0_{l\times l}$;
- $\text{MSoftmax}(\boldsymbol{Z}, \boldsymbol{M})$ is stable (no abrupt changes) as $\boldsymbol{M} \to 0_{l\times l}$

C Application in Sequence Compression

Autoencoders are a common choice for unsupervised learning. Its goal is to compress the input features into a lower-dimensional latent representation, by learning a pair of encoder and decoder that compress and reconstruct the input respectively so that the reconstruction loss is minimized. However, this latent representation has a fixed shape, which is suboptimal when the amount of information in the input have a large variance across samples.

For example, byte-level language modeling (Pagnoni et al., 2024; Wang et al., 2024) and latent space language modeling (team et al., 2024) both involve converting a long, low-level sequence into a shorter sequence of high-level latent patches to allow for efficient processing. However, deciding the size and position of each patch is non-trivial. A potential solution is to use an autoencoder that consists of two sequence models (e.g. transformers) and apply ReM on the encoded sequence to induce token level sparsity (i.e., mask out entire tokens). Then, the non-zero token embeddings in the resulting sequence can be used as input embeddings to a large, latent-space language model.