Merging Feed-Forward Sublayers for Compressed Transformers

Anonymous ACL submission

Abstract

With the rise and ubiquity of larger deep learning models, the need for high-quality compression techniques is growing in order to deploy these models widely. The sheer parameter count of these models makes it difficult to fit them into the memory constraints of different hardware. In this work, we present a novel approach to model compression by *merging* similar parameter groups within a model, rather than pruning away less important parameters. Specifically, we select, align, and merge separate feed-forward sublayers in Transformer models, and test our method on language modeling, image classification, and machine translation. With our method, we demonstrate performance comparable to the original models while combining more than a third of model feed-forward sublayers, and demonstrate improved performance over a strong layer-pruning baseline. For instance, we can remove over 21% of total parameters from a Vision Transformer, while maintaining 99% of its original performance. Additionally, we observe that some groups of feed-forward sublayers exhibit high activation similarity, which may help explain their surprising mergeability.

1 Introduction

Recent advances in deep learning have been marked by large, pre-trained models in order to achieve state-of-the-art performance. With this trend towards growing parameter counts, more high-quality compression techniques are needed that balance compression effectiveness and model performance. These techniques help facilitate model use across a variety of inference settings and hardware availability.

Much of the prior work in model compression has built upon on distillation, quantization, and pruning techniques (Hinton et al., 2015; Fiesler et al., 1990; LeCun et al., 1989). Prior work on pruning has introduced many techniques identifying regions of parameters that can be removed from the model without drastically changing performance. These techniques target individual neurons or general regions of a model—like attention heads, parameter chunks, or even entire layers. (Voita et al., 2019; Lagunas et al., 2021; Sajjad et al., 2023). However, while "unimportant" features are targeted for pruning techniques, we can also target "redundant" features for compression. There has been far less focus on compression methods that target redundancy within a model.

When targeting redundant features for compression, we can turn to *merging* sets of similar parameters rather than pruning them. Relatedly, the research area of model merging has explored merging parameters from two or more separate models in order to combine their functionalities into a single model (Goddard et al., 2024; Yang et al., 2024a). In our case, we can imagine extending parameter merging to merge *sublayers* within one model, rather than just separate models.

To this end, we propose a novel compression method that aligns, merges, and ties separate feed-forward (FF) sublayers within Transformer architectures (Vaswani et al., 2017). We target FF sublayers in particular due to their large parameter count and easy mergeability. Through our testing, we find that these groups of FF sublayers are notably compressible via merging, giving rise to a simple and surprisingly effective framework applicable to a variety of existing pre-trained models.

We highlight the contributions of our work:

1. We propose a novel model compression method inspired by recent work in model merging. This approach is orthogonal to popular compression methods like quantization.
2. Across three different Transformer-based models, namely GPT-2, ViT, and a machine translation model, we show that merging over one-third of feed-forward sublayers and fine-

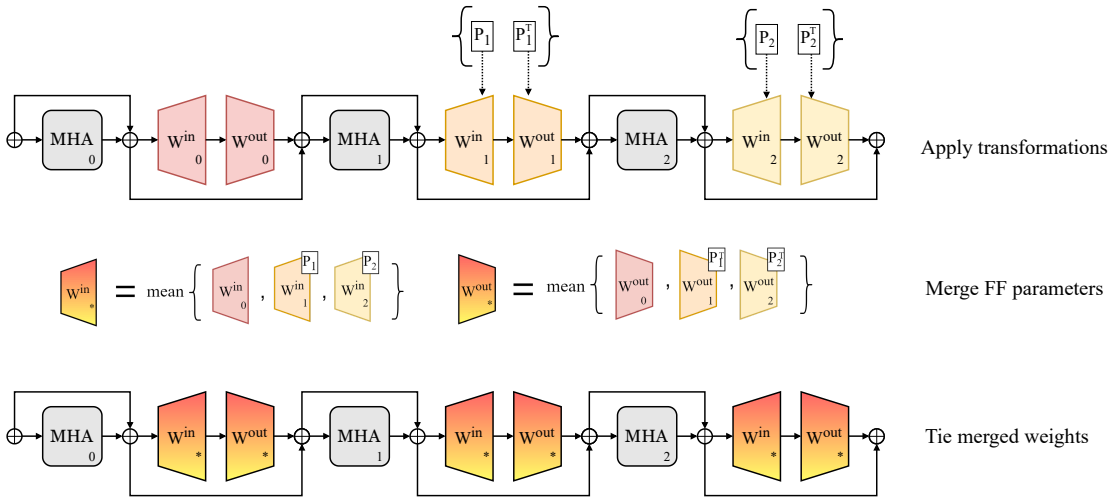


Figure 1: Overview of the feed-forward alignment and merging algorithm used to compress models in an example three layers of a Transformer. Multi-headed attention is abbreviated to MHA, feed-forward sublayers are depicted with W^{in} and W^{out} weights, and Add&Norm operations are depicted with $\bigoplus$, connected by arrows indicating residual connections.[1] Permutation transformation matrices are shown as P_i. Our method includes a permutation finding step, applying the transformations, merging transformed parameters, and finally tying the merged parameters. By merging and tying k feed-forwards, we can reduce the model size by $k - 1$ feed-forward sublayers.

tuning the resulting model can achieve performance comparable to the original models.

3. To explore the surprising effectiveness of merging, we compare different feed-forward outputs from the same model, and find regions with highly similar activations. These same patterns do not occur in attention outputs.

4. We release an easily extensible toolkit for our compression method at anonymous_code.

2 Related Work

In this section, we review prior work related to weight tying for efficient models, and work related to pruning and redundancy. We also summarize major compression techniques in Table 1, and compare them to our merging-based approach.

2.1 Weight tying for smaller models

Prior work on weight tying has largely focused on training models from scratch with specific tying schemes. Tying input and output embedding layers helps cap total parameter count, but more importantly provides important gradient sharing signal for better generalization in language generation tasks (Press and Wolf, 2017; Inan et al., 2017). In the case of non-embedding layers in Transformers, prior work has explored numerous weight tying patterns for training new models (Dehghani et al., 2019; Reid et al., 2021; Takase and Kiyono, 2023). Liu et al. (2024) use heavy weight sharing between Transformer layers at initialization to achieve state-of-the-art sub-billion parameter language models. Pires et al. (2023) specifically tie widened FF sublayers at initialization and train machine translation (MT) models that outperform standard Transformer MT models. In our work, we instead start from a pre-trained model, and then use weight sharing as a tool to reduce the overall parameter count.

2.2 Pruning and redundancy

Prior work has explored different aspects of redundancy between Transformer components, and suggested several techniques to reduce or exploit this phenomenon. Dalvi et al. (2020) use centered kernel alignment (CKA) to show layer redundancy in BERT and XLNet, and use correlation clustering to find and remove redundant sets of neurons. Men et al. (2024); Gromov et al. (2024) propose removing entire Transformer layers in deep, decoder-only language models to achieve inference speedups at a small performance drop. Li et al. (2024) propose a compression method for sparsely-activated

[1] This diagram shows a Post-LN Transformer, but our method easily applies to Pre-LN Transformers as well.

[2] Quantization can improve batch throughput during inference, which can result in run time savings, but it generally does not improve inference speed at a constant batch size.

	Motivation	Training Required	Run Time Savings
Quantization	reduce precision	No	No[2]
Pruning	remove unimportant parameters	generally fine-tuning	Depends
Distillation	train smaller student from teacher	Yes	Yes
Merging	combine redundant parameters	fine-tuning	No

Table 1: A summary and comparison of different compression methods, including merging.

mixture-of-expert (SMoE) models that draws from model merging work to compress some experts in large SMoE models. Our method extends a similar approach to a much wider set of models.

3 Merging Feed-Forward Sublayers

In this section, we discuss feed-forward sublayers as a merging target, explain permutation-based neuron alignment, and describe our compression method.

3.1 Targeting feed-forward sublayers

We focus our interest on Transformer FF sublayers for several reasons. Firstly, these sublayers constitute around two-thirds of non-embedding parameters in Transformer encoder or decoder models. Compressing these parameters can result in substantial overall savings in a model. Secondly, the parameterization of FF sublayers is far simpler than the other major sub-block of a Transformer layer, namely multi-headed attention (MHA). This structural simplicity makes it a good candidate for merging-based compression approaches.

Beyond these practical considerations, prior work establishes several properties of Transformer FF sublayers that make them good candidates for compression via merging. Li et al. (2023) show that they can be very sparsely activated, where non-zero FF activations can be as low as 3-5%. Other work has demonstrated evidence that adjacent LayerNorm and FF blocks, in both Post- and Pre-LN architectures, results in some weakening of the contextualization effects of FF sublayers (Kobayashi et al., 2024). The authors allude to redundancy in Transformer FF processing due to this interaction. Finally, Pires et al. (2023) train Transformer-based translation models with only one widened and tied encoder FF block with experimental success.

3.2 Background on permutation-based neuron alignment

We propose a merging technique that combines several similar sublayers into a single parameter set. Our merging technique is inspired by prior work in permutation symmetries of neurons (Li et al., 2015). This technique has been used in studying convergent learning between models, as well as performing model merging between two or more separate models (Tatro et al., 2020; Entezari et al., 2022; Ainsworth et al., 2023).

Permutation-based neuron alignment techniques seek to find an optimal ordering of neurons in one layer that more closely matches the ordering of neurons from another layer, without changing the its output. Given two layers to align, we compute a forward pass through both using exemplar data in order to collect activations. The layers are generally corresponding parameters from different models. This results in two activation sets $X_\alpha, X_\beta \in \mathbb{R}^{n \times d}$, where n is the number of example data points, and d is the model dimension.

To determine corresponding neurons from the activations, we compute cross-correlation C, in line with prior work (Li et al., 2015). μ represents mean vectors, and σ standard deviation vectors.

$$C = \frac{\mathbb{E}\left[\left(X_\alpha - \mu(X_\alpha)\right)^T \left(X_\beta - \mu(X_\beta)\right)\right]}{\sigma(X_\alpha)\sigma(X_\beta)} \quad (1)$$

The resulting matrix $C \in \mathbb{R}^{d \times d}$ reflects how each neuron j in X_α correlates with each neuron k in X_β. To find the neuron alignment that maximizes total correlation, we solve the following optimization problem, where Π_d is the space of all permutations of length d (Li et al., 2015; Tatro et al., 2020):

$$\pi^* = \max_{\pi \in \Pi_d} \sum_{j=1}^{d} C(j, \pi(j)) \quad (2)$$

This problem is a case of the Linear Assignment Problem, and we solve for π^* using the Jonker-Volgenant algorithm implementation provided by `scipy` (Crouse, 2016).

3.3 Combining feed-forward sublayers

Now, with the appropriate background, we describe our compression method. For our method, we first

assume that we have some predetermined number of feed-forward sublayers k that we want to merge. This number can be inferred given a overall parameter reduction ratio, or set otherwise. In summary, our compression method aligns the ordering of the neurons between the multiple feed-forward sublayers in order to merge them.

Given a window of k adjacent feed-forward sublayers, we compute a forward pass using a subset of data in order to compute features for each feed-forward hidden state. In other words, for Transformer FF sublayer $x^{\text{out}} = W^{\text{out}}\phi(W^{\text{in}}x^{\text{in}} + b^{\text{in}}) + b^{\text{out}}$, we obtain features just before the ϕ activation. We consider only the neurons just *after* W^{in} because prior work has shown that to reorder the input to W^{in} and output of W^{out} requires permuting many additional weights due to the residual connections in order to maintain functional equivalence (Verma and Elbayad, 2024). For each of the k feed-forward sublayers, we collect features $X_i \in \mathbb{R}^{n \times d} \;\; i \in [0, k-1]$, where n is the number of tokens or patches processed, and d is the feed-forward dimension.[3]

We designate the first feed-forward sublayer of the set to be an "anchor", and compute the permutation-finding algorithm on each pair of features where one index is always the anchor. In other words, for each sublayer $i \in [1, k-1]$, we have inputs X_0 and X_i, and find π_i using the permutation finding algorithm from Section 3.2.

After converting function π_i to its corresponding permutation matrix P_i, we transform the $k-1$ non-anchor feed-forward sublayers. We then average these k FF sublayers, and replace each of them with their average, as in Equations 3–6. Finally, we tie these weights so that in memory they appear as just one sublayer, effectively removing the parameters from $k-1$ FF sublayers.

$$W^{\text{in}*} = \frac{1}{k}\left(W_0^{\text{in}} + \sum_{i=1}^{k-1} P_i W_i^{\text{in}}\right) \tag{3}$$

$$b^{\text{in}*} = \frac{1}{k}\left(b_0^{\text{in}} + \sum_{i=1}^{k-1} P_i b_i^{\text{in}}\right) \tag{4}$$

$$W^{\text{out}*} = \frac{1}{k}\left(W_0^{\text{out}} + \sum_{i=1}^{k-1} W_i^{\text{out}} P_i^T\right) \tag{5}$$

$$b^{\text{out}} = \frac{1}{k}\left(\sum_{i=0}^{k-1} b_i^{\text{out}}\right) \tag{6}$$

[3]The layer indices reflect local index within the set of k versus global layer index.

3.4 Selecting sublayers to merge

In selecting the k adjacent feed-forward sublayers to merge, we take a sliding window approach. For all starting layer indices from 0 to $(N_{\text{layers}} - 1) - k$, we apply the method outlined in Section 3.3, and evaluate the resulting compressed model on a validation set.

Although we propose to test each potential window, in reality, the cost of computing permutations and parameter arithmetic is low. The largest costs in each iteration is computing features and testing candidates. However, we only compute features *once* despite testing $N_{\text{layers}} - k$ models, because one forward pass through the exemplar data is sufficient for creating all necessary correlation matrices. The best candidate is the one with the highest post-merge evaluation score. We note that there may be other possible selection heuristics in this setting,.

Finally, we follow our merging procedure with recovery fine-tuning to quickly heal performance on the downstream task. We include an algorithm for our selection method in Algorithm 1.

Algorithm 1 Feed-Forward Sublayer Merge

Input: Model parameters θ_{in}, collected features $\{X_i\}_{i=0}^{N_{\text{layers}}-1}$, batched fine-tuning data D_{ft}
Input constants: k, N_{layers}, MAXUPDATES
Initialize: θ_{selected}, BESTSCORE $\leftarrow 0$
for $i = 0$ **to** $(N_{\text{layers}} - 1) - k$ **do**
  $\theta_{\text{merged}} \leftarrow$ COMPRESS$(\theta_{\text{in}}, \{X_i\}_{i=0}^{N_{\text{layers}}-1}, k)$
  if EVAL(θ_{merged}) > BESTSCORE **then**
    $\theta_{\text{selected}} \leftarrow \theta_{\text{merged}}$
  end if
end for
for $i = 0$ **to** MAXUPDATES **do**
  $\theta_{\text{selected}} \leftarrow$ UPDATE$(\theta_{\text{selected}}, D_{\text{ft}}(i))$
end for
Output: θ_{selected}

4 Experimental Setup

For testing the extensibility of our method, we apply our compression method to several different Transformer-based models. Specifically, we use GPT-2 (Radford et al., 2019), the Vision Transformer (ViT) (Dosovitskiy et al., 2020), and a Transformer-based machine translation model from OPUS-MT (Tiedemann and Thottingal, 2020). We select this variety of models in order to cover a diversity of model types (decoder-only, encoder, encoder-decoder) and different modalities.

For each setting, we list the model used, the example data for computing alignments, and finally the data used for recovery fine-tuning and evaluation. Additional fine-tuning hyperparameters are included in Appendix A, and data details in Appendix B.

4.1 Language modeling

For our experiments, we use GPT-2 Large, which has 36 layers, a feed-forward dimension of 5120, and is trained on English text (Radford et al., 2019). For computing example activations, we use 10k tokens from the validation set of the English Wikitext103 dataset (Merity et al., 2017). Finally, we use the train and test sets from the Wikitext103 for fine-tuning and evaluation, respectively.

Unlike the other two tasks, the pre-training data for GPT-2 is not publicly available, so we use Wikitext103 training data for fine-tuning. Due to this discrepancy, our uncompressed GPT-2 baseline is also fine-tuned on Wikitext103 train. Because we have access to the training data for our machine translation and ViT models, we do not provide a fine-tuned baseline for those as the data we use already appears in their original training data.

We fine-tune our GPT-2 models for up to 100k steps with batches of 2048 tokens. We select the best model based on validation perplexity and report average test perplexity with a sliding window of 512 tokens.

4.2 Image classification with ViT

We use a vision transformer (ViT) for our image classification experiments, with resolution of 224x224, and patch size of 16x16 (Dosovitskiy et al., 2020). ViT is a 12-layer Transformer encoder model pre-trained on ImageNet-21k, and subsequently fine-tuned on ImageNet-1k. ImageNet-1k is a classification task where images belong to one of 1000 categories (Russakovsky et al., 2015). For computing activations, we use 10k patches from the ImageNet-1k validation set. Evaluation results are computed on original validation labels.

We fine-tune our ViT models on ImageNet-1k train for up to 50k steps with a batch size of 128, and report accuracy.

4.3 Machine translation

For our experiments on machine translation, we use a 12-layer Chinese-English Transformer-based translation model from an OPUS-MT release (Tiedemann and Thottingal, 2020). For computing activations, we use 10k tokens from the Tatoeba validation set[4] (Tiedemann, 2020). For fine-tuning, we use the original training data released by the Tatoeba translation challenge, sourced from OPUS (Tiedemann, 2012). We apply our method to both the encoder and decoder separately, constituting two anchors. However, we search windows in sync, meaning that the same window from the encoder and decoder are merged, but separately.

We fine-tune our translation models for up to 100k steps with a batch size of 64 sentences. We use `sacrebleu` to compute BLEU scores for evaluation (Papineni et al., 2002; Post, 2018).

4.4 Layer pruning baseline

Recent work on structured pruning of Transformers has seen many methods presenting ways to remove full layers from a model and then optionally fine-tune the compressed model (Men et al., 2024; Gromov et al., 2024; Yang et al., 2024b). We focus on a structured pruning baseline as many unstructured pruning methods do not actually realize compression unless they achieve 1) high sparsity ratios and 2) use specialized sparse libraries to store sparse weights. On the other hand, our method easily realizes compression due to weight tying.

Many layer-pruning methods rely on similarity measures to choose a set of adjacent layers to prune. However, we forgo any specific similarity techniques and instead choose the best subset after evaluation much like our own technique, via a sliding window. After selecting the best pruned model, we then fine-tune the model with the same specifications as our method. In all, this encapsulates a strong, structured pruning baseline that generalizes many layer-pruning based techniques.

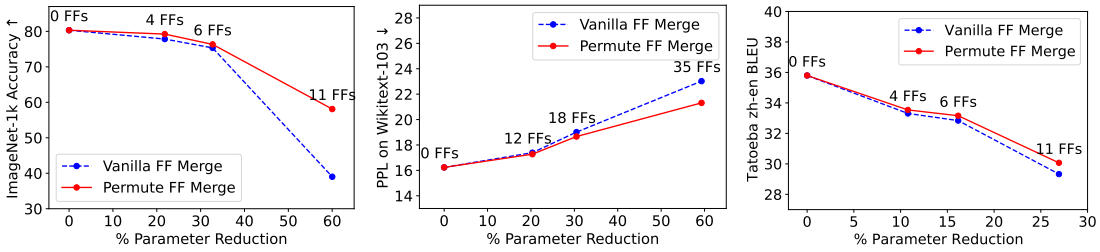
5 Results

5.1 Merging feed-forward sublayers across compression ratios

We evaluate our compression method on image classification using ViT, language modeling using GPT-2, and machine translation using an OPUS-MT zh-en model, and report our results in Figure 2. We report results at 1/3, 1/2 and $(n-1)/n$ feed-forward sublayers removed, in order to test our method at different overall compression ratios.[5] We

[4] counted on the source side

[5] We note that the OPUS-MT ratios trends are different due to the enc-dec architecture.



(a) ViT (b) GPT-2 (c) OPUS-MT

Figure 2: Results across all three tasks depicting compression versus performance results. We include results from our main method, labeled as Permute FF Merge, as well as our method without permutation alignment, depicted as Vanilla FF Merge. We note that our method retains almost complete performance at one-third of feed-forward sublayers removed, across all tasks, and continues to retain high performance at one-half of FF sublayers removed.

also report results from our compression method without the permutation step, labeled as "Vanilla."

From our results, we see that even up to 1/2 of FF sublayer parameters removed, which is over 30% in parameter reduction for ViT and GPT-2,[6] our method can retain high performance, similar to the base model. At 1/3 of FF sublayers removed, performance is almost identical to the original model, resulting in only a 1% accuracy drop in ViT, 1 PPL increase in GPT-2, and 2 BLEU drop in the translation model. Full numerical results can be found in Appendix C. Prior work suggests in this sub-billion parameter regime, smaller models are more difficult targets of compression methods (Ashkboos et al., 2024).

Our findings also hold across all three of our tasks tested, suggesting that our method generalizes to different types of models. Additionally, we can notice that permutation-based compression is consistently better compared to no-permute vanilla baselines, demonstrating the effectiveness of aligning features before merging. This effectiveness is more pronounced at larger numbers of FF sublayers removed. In summary, our results show that 1) post-training weight tying is a simple and effective compression method and 2) permutation-based alignment of these shared weights can improve final compression performance.

In Figure 3, we compare our method at 1/3 and 1/2 FFs removed to our layer-pruning baseline.[7] We drop layers to attempt to match the reduction ratios of our own methods, constituting 1/6 and 1/3 of layers dropped for all three models. However, since we cannot match exact ratios, we plot the exact parameter reduction ratios and performance, and compare. As seen in the figure, our method consistently matches or outperforms the layer-dropping method. This comparison confirms that merging is a competitive alternative to strong pruning-based methods for model compression.

5.2 Choice of merged sublayers

In our merging algorithm, we choose which layers to merge by computing performance over sliding windows of k indices. For each of our model/task pairs, we plot the pre-tuning performance of the merging algorithm on 1/3 of FF sublayers dropped across all windows, to observe their differences. Results are shown in Figure 4. Before tuning, it appears that the choice of layers seems to be important, resulting in different performance.

However, these differences reduce once recovery fine-tuning is performed. To see this, we randomly select 3 sets of k consecutive layers for each of our tasks, and apply recovery fine-tuning to these compressed models. In Table 2, we observe that all models achieve similar performance after fine-tuning. Nevertheless, the choice of layers might be important if non-adjacent merges are allowed; this is potential future work.

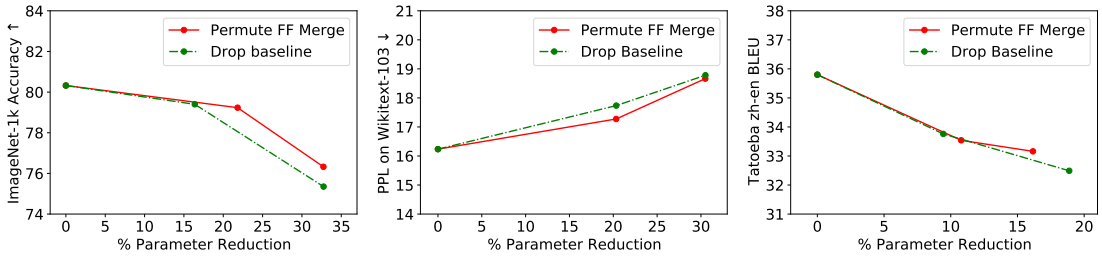
5.3 Choice of anchor layer

In addition to analyzing the subset of layers to merge, we also wish to understand the sensitivity of our merging compression method to the choice of anchor layer for our alignment step. In section 3.3, we choose the first feed-forward sublayer in the sequence to serve as the reference, and compute permutations aligning the following sublayers to

[6] We include embedding parameters in all % parameter reduction and compression ratio calculations.

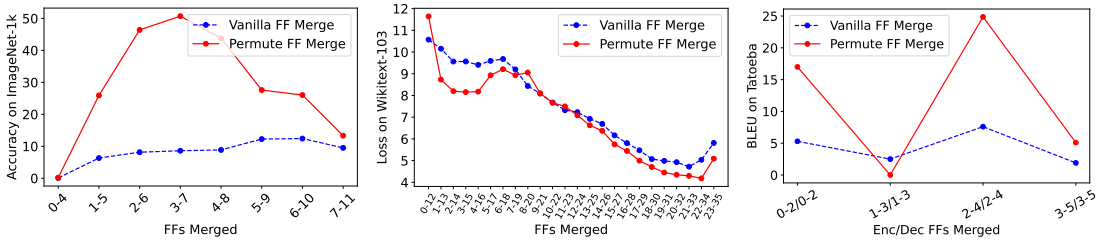
[7] These reduction ratios reflect ratios found in layer-pruning literature.

[8] We display loss on Wikitext-103 for visibility.



(a) ViT (b) GPT-2 (c) OPUS-MT

Figure 3: Results across all three tasks depicting compression versus performance for our method and a strong layer-dropping baseline method. We perform layer dropping for 1/6 and 1/3 of layers dropped, and fine-tune the best pre-tuned set of dropped layers for all sliding windows. Across the parameter reduction range shown, our merging-based compression method outperforms or matches layer-dropping across the three tasks.



(a) ViT (b) GPT-2 (c) OPUS-MT

Figure 4: Performance curves over different ranges of merged feed-forward sublayers representing 1/3 FFs removed. Across all three tasks, there are clear ranges of merged sublayers that retain more performance when merged.[8]

	ViT Accuracy(%) ↑	GPT-2 PPL ↓	OPUS-MT BLEU ↑
Best pre-tune	79.2	17.3	33.5
Random 1	79.5	18.3	33.9
Random 2	78.5	17.1	33.8
Random 3	78.9	17.3	33.1

Table 2: Results comparing our compression method @ 1/3 of feed-forward sublayers removed with different sublayer groups. We include three random consecutive selections of sublayers, excluding the original selection.

	ViT Accuracy(%) ↑	GPT-2 PPL ↓	OPUS-MT BLEU ↑
Anchor First	79.2	17.3	33.5
Anchor Middle	79.5	17.4	33.4
Anchor Last	79.0	17.4	33.5

Table 3: Results comparing our compression method with 1/3 of feed-forward sublayers removed, but with different anchor locations.

this reference. Here, we additionally consider using either the *last* of the sequence, or the *middle* of the sequence, and report results in our 1/3 feed-forward merge setting in Table 3.

Given the similar results across settings, our merging approach is robust to the choice of reference or anchor layer, enhancing the reliability of our permutation-based alignment method to find corresponding features for a useful merge.

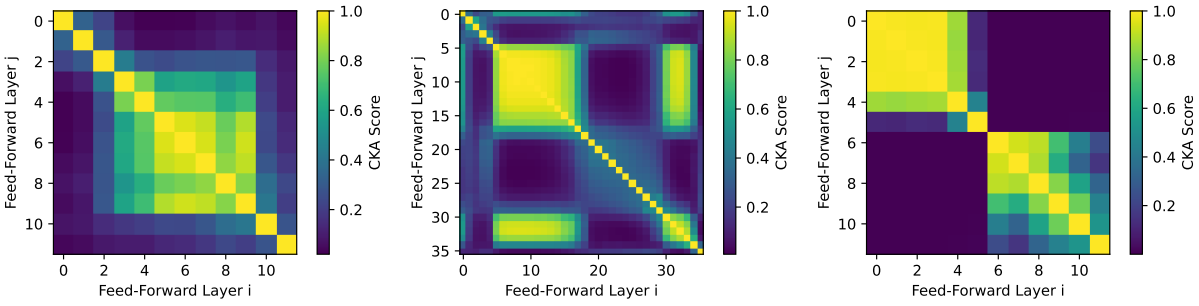
5.4 Additional compression via quantization

While our compression method focuses on reducing model size via parameter sharing, quantization can also reduce the overall storage needed for a model via reducing parameter precision. If our method performs orthogonally to state-of-the-art quantization, both methods may be used together for additional storage savings. We experiment with the LLM.int8() quantization method due to its effectiveness and widespread adoption (Dettmers et al., 2022). We quantize our models after removing 1/3 of FF sublayers, and report results in Table 4.

Combining our method with quantization provides even smaller compression ratios, while retaining high performance. Coupling quantization with additional compression, like our method, helps to realize compression ratios like 20% when considering total model storage complexity.

		Our Method		+LLM.int8()	
Model	Metric	Compression	Performance	Compression	Performance
ViT	Accuracy(%) ↑	78%	79.2	20%	79.2
GPT-2	PPL ↓	80%	17.3	22%	17.3
OPUS-MT	BLEU ↑	89%	33.5	51%	33.5

Table 4: Compression results across three tasks, before and after additional compression via quantization. In this case, compression is measured in terms of total model storage complexity (disk space) instead of parameter count.



(a) ViT (b) GPT2 (c) OPUS-MT

Figure 5: CKA plots of feed-forward sublayer hidden states across three different models. In all three settings, we see clear regions of high similarity between different FF layers. We do not compare between encoder and decoder feed-forward sublayers in the Translation model due the differences in token inputs.

5.5 Similarity trends across feed-forward sublayers

Given the success of simply aligning and merging adjacent feed-forward sublayers for compression, we look further into possible signs of redundancy in their representations, as alluded to in previous work (Pires et al., 2023; Kobayashi et al., 2024).

To this end, we compare outputs between FF sublayers within the same models. Across our three tasks, we use 10k tokens or patches from task validation sets to compute a set of output states from all feed-forward sublayers. Then, we use Centered Kernel Alignment (CKA) to compute their similarity. CKA is a state-of-the-art method for comparing the similarity between neural network activations (Kornblith et al., 2019). We plot similarity values for all pairwise interactions between FF sublayers in all three of our model types, shown in Figure 5.

We notice that across all three model/task pairs, clear regions of high similarity between FF outputs can be observed, despite FF sublayers being interleaved with multi-headed attention sublayers. We note that similar behavior is not seen in attention sublayers, as seen in Appendix D. While prior work has shown similarities between the outputs of adjacent *full* Transformer layers, this similarity can be explained in part to the residual computations that add the prior sublayer output to the current sublayer output (Kornblith et al., 2019; Dalvi et al., 2020). However, here we isolate the FF outputs from the stream of residual computations, before this output is added back to its input, making the observed similarity more surprising due to the greater independence between FF computations.

6 Conclusion

In this work, we propose a novel compression method that applies to Transformer models via merging and tying adjacent sets of FF sublayers. Our method serves as an alternative to existing compression approaches, and opens possibilities of future methods that examine the use of parameter merging and weight tying as a post-training compression technique. We demonstrate our method's extensibility across diverse tasks, and show that it helps retain high performance even after removing 1/2 of FF sublayers, and outperforms a strong layer pruning baseline. Finally, we find that several FF sublayers activate very similarly despite being separated by attention sublayers, which may be related to their surprising mergeability.

7 Limitations

This merging-based compression method sits between many unstructured pruning methods and structured pruning methods, where the former generally does not result in speed-up or easily realized compression, but the latter can more easily lead to both speed-up and easily realized compression. Given that this work does lead to easily realized compression, but does not create an inference speed-up, this is a main limitation of our work.

Additionally, our method is designed and tested on models that use a Transformer-based architecture. While weight-tying and neuron alignment may apply straightforwardly to other architectures, we do not test this, which constitutes another limitation of this work.

References

Samuel Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. 2023. Git re-basin: Merging models modulo permutation symmetries. In *The Eleventh International Conference on Learning Representations*.

Saleh Ashkboos, Maximilian L Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. 2024. Slicegpt: Compress large language models by deleting rows and columns. In *The Twelfth International Conference on Learning Representations*.

David F. Crouse. 2016. On implementing 2d rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4):1679–1696.

Fahim Dalvi, Hassan Sajjad, Nadir Durrani, and Yonatan Belinkov. 2020. Analyzing redundancy in pretrained transformer models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4908–4926, Online. Association for Computational Linguistics.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *International Conference on Learning Representations*.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. Gllm.int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. 2022. The role of permutation invariance in linear mode connectivity of neural networks. In *International Conference on Learning Representations*.

Emile Fiesler, Amar Choudry, and H John Caulfield. 1990. Weight discretization paradigm for optical neural networks. In *Optical interconnections and networks*, volume 1281, pages 164–173. SPIE.

Charles Goddard, Shamane Siriwardhana, Malikeh Ehghaghi, Luke Meyers, Vlad Karpukhin, Brian Benedict, Mark McQuade, and Jacob Solawetz. 2024. Arcee's mergekit: A toolkit for merging large language models. *arXiv preprint arXiv:2403.13257*.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A Roberts. 2024. The unreasonable ineffectiveness of the deeper layers. *arXiv preprint arXiv:2403.17887*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *stat*, 1050:9.

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2017. Tying word vectors and word classifiers: A loss framework for language modeling. In *International Conference on Learning Representations*.

Goro Kobayashi, Tatsuki Kuribayashi, Sho Yokoi, and Kentaro Inui. 2024. Analyzing feed-forward blocks in transformers through the lens of attention maps. In *The Twelfth International Conference on Learning Representations*.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. 2019. Similarity of neural network representations revisited. In *International conference on machine learning*, pages 3519–3529. PMLR.

François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M Rush. 2021. Block pruning for faster transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10619–10629.

Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. *Advances in neural information processing systems*, 2.

Pingzhi Li, Zhenyu Zhang, Prateek Yadav, Yi-Lin Sung, Yu Cheng, Mohit Bansal, and Tianlong Chen. 2024. Merge, then compress: Demystify efficient smoe with hints from its routing policy. In *The Twelfth International Conference on Learning Representations*.

Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John Hopcroft. 2015. Convergent learning: Do different neural networks learn the same representations? In *Proceedings of the 1st International Workshop on Feature Extraction: Modern Questions and Challenges at NIPS 2015*, volume 44 of *Proceedings of Machine Learning Research*, pages 196–212, Montreal, Canada. PMLR.

Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank J Reddi, Ke Ye, Felix Chern, Felix Yu, Ruiqi Guo, et al. 2023. The lazy neuron phenomenon: On emergence of activation sparsity in transformers. In *The Eleventh International Conference on Learning Representations*.

Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, et al. 2024. Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. In *Forty-first International Conference on Machine Learning*.

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2024. Shortgpt: Layers in large language models are more redundant than you expect. *Preprint*, arXiv:2403.03853.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *International Conference on Learning Representations*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Telmo Pires, António Vilarinho Lopes, Yannick Assogba, and Hendra Setiawan. 2023. One wide feedforward is all you need. In *Proceedings of the Eighth Conference on Machine Translation*, pages 1031–1044, Singapore. Association for Computational Linguistics.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163.

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

Machel Reid, Edison Marrese-Taylor, and Yutaka Matsuo. 2021. Subformer: Exploring weight sharing for parameter efficiency in generative transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4081–4090, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2023. On the effect of dropping layers of pre-trained transformer models. *Computer Speech & Language*, 77:101429.

Sho Takase and Shun Kiyono. 2023. Lessons on parameter sharing across layers in transformers. In *Proceedings of The Fourth Workshop on Simple and Efficient Natural Language Processing (SustaiNLP)*, pages 78–90, Toronto, Canada (Hybrid). Association for Computational Linguistics.

Norman Tatro, Pin-Yu Chen, Payel Das, Igor Melnyk, Prasanna Sattigeri, and Rongjie Lai. 2020. Optimizing mode connectivity via neuron alignment. *Advances in Neural Information Processing Systems*, 33:15300–15311.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in OPUS. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 2214–2218, Istanbul, Turkey. European Language Resources Association (ELRA).

Jörg Tiedemann. 2020. The Tatoeba Translation Challenge – Realistic data sets for low resource and multilingual MT. In *Proceedings of the Fifth Conference on Machine Translation*, pages 1174–1182, Online. Association for Computational Linguistics.

Jörg Tiedemann and Santhosh Thottingal. 2020. OPUS-MT – building open translation services for the world. In *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation*, pages 479–480, Lisboa, Portugal. European Association for Machine Translation.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Neural Information Processing Systems*.

Neha Verma and Maha Elbayad. 2024. Merging text transformer models from different initializations. *Preprint*, arXiv:2403.00986.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy. Association for Computational Linguistics.

Enneng Yang, Li Shen, Guibing Guo, Xingwei Wang, Xiaochun Cao, Jie Zhang, and Dacheng Tao. 2024a. Model merging in llms, mllms, and beyond: Methods, theories, applications and opportunities. *Preprint*, arXiv:2408.07666.

Yifei Yang, Zouying Cao, and Hai Zhao. 2024b. Laco: Large language model pruning via layer collapse. *arXiv preprint arXiv:2402.11187*.

A Fine-tuning details

A.1 GPT-2

Hyperparameter	Value
Start LR	5e-5
LR Schedule	`inv_sqrt`
fp16	True
batch size	2
n_steps	100K

Table 5: Hyperparameters used for GPT-2 fine-tuning

A.2 ViT

Hyperparameter	Value
Start LR	5e-5
LR Schedule	`lin_decay with min`
decay_steps	20K
Min LR	1e-6
fp16	True
batch size	128
n_steps	50K

Table 6: Hyperparameters used for ViT fine-tuning

A.3 Machine Translation

We select our best model using validation BLEU, computed on a 2000 instance subset of the full Tatoeba validation set.

Hyperparameter	Value
Start LR	5e-5
LR Schedule	`inv_sqrt`
fp16	True
batch size	64
n_steps	100K

Table 7: Hyperparameters used for OPUS-MT fine-tuning

B Dataset details

We report the dataset statistics for our evaluations and training data used in this work in Table 8. For fine-tuning data, we note that updates reported in Appendix A give a better idea of data usage rather than the training counts provided here.

Dataset	Train	Validation	Test
ImageNet-1k	12,281,167	50,000	-
Wikitext-103	1,801,350	3,760	4,358
OPUS/Tatoeba	41,649,946	43,074	10,389

Table 8: The number of instances used in each fine-tuning and evaluation datasets. Instances are image for ImageNet, lines of text for Wikitext-103, and bitext pairs for OPUS/Tatoeba.

C Full Results at varying compression ratios

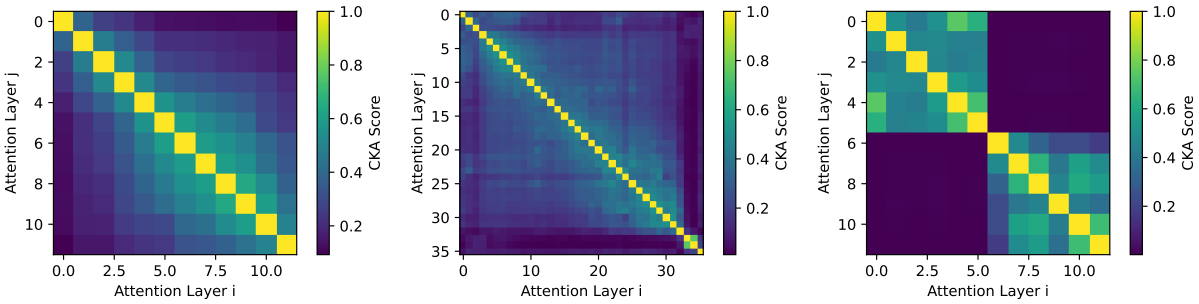
We report our full results across compression ratios in Table 9.

D Attention Layer Similarity

We compute CKA similarity between all attention sublayer pairs, using the same 10k tokens or patches from our CKA results on FF sublayers. The features are from the output of the linear layer just after the dot-product attention computation. Results appear in Figure 6.

Model	Metric	Merged Indices	FFs Removed	Vanilla	Permute
ViT	Accuracy (%) ↑	–	0/12	80.3	80.3
		3-7	4/12	77.8	79.2
		4-10	6/12	75.3	76.3
		0-11	11/12	39.0	58.1
GPT-2	PPL ↓	–	0/36	16.16	16.16
		22-34	12/36	17.39	17.27
		16-34	18/36	19.01	18.66
		0-35	35/36	23.02	21.31
OPUS-MT	BLEU ↑	–	0/12	35.8	35.8
		2-4/2-4	4/12	33.3	33.5
		0-3/0-3	6/12	32.8	33.2
		0-5/0-5	11/12	29.3	30.1

Table 9: Full numerical results on compression results at 1/3 FF sublayers removed, 1/2 FF sublayers removed, and $(n-1)/n$ FF sublayers removed. Original, uncompressed models are included in the first row of results for each model, indicated by 0 FFs removed and no merged indices.



(a) ViT (b) GPT2 (c) OPUS-MT

Figure 6: CKA plots of multi-headed self-attention sublayer activations across three different trained models. Attention activations are largely dissimilar from each other across model types. We do not compare between encoder and decoder attention sublayers in the translation model due the differences in token inputs.