CHAI: Clustered Head Attention for Efficient LLM Inference

Saurabh Agarwal [1] **Bilge Acun** [2] **Basil Hosmer** [2] **Mostafa Elhoushi** [2] **Yejin Lee** [2] **Shivaram Venkataraman** [1]
Dimitris Papailiopoulos [1] **Carole-Jean Wu** [2]

Abstract

Large Language Models (LLMs) with hundreds of billions of parameters have transformed the field of machine learning. However, serving these models at inference time is both compute and memory intensive, where a single request can require multiple GPUs and tens of Gigabytes of memory. Multi-Head Attention is one of the key components of LLMs, which can account for over 50% of LLMs memory and compute requirement. We observe that there is a high amount of redundancy across heads on which tokens they pay attention to. Based on this insight, we propose Clustered Head Attention (CHAI). CHAI combines heads with a high amount of correlation for self-attention at runtime, thus reducing both memory and compute. In our experiments, we show that CHAI is able to reduce the memory requirements for storing K,V cache by up to 21.4% and inference time latency by up to 1.73× without any fine-tuning required. CHAI achieves this with a maximum 3.2% deviation in accuracy across 3 different models (i.e. OPT-66B, LLAMA-7B, LLAMA-33B) and 5 different evaluation datasets.

1. Introduction

LLMs have demonstrated remarkable performance on language modelling tasks ranging from question answering, text summarizing, language translation. However, such performance has been achieved by scaling models to trillions of parameters, and existing works (Hoffmann et al., 2022; Touvron et al., 2023a; Kaplan et al., 2020) show that increasing the model size may lead to even higher model quality.

Inference on LLMs introduce several new challenges. Beyond just the quadratic computation cost of self-attention (Vaswani et al., 2017) with increasing context and large model sizes, LLMs also store intermediate Key (K) and Value (V) pairs for subsequent next word prediction. This K,V caching introduces additional memory related challenges as K,V cache size increases with increase in sequence length. The architecture of widely used LLMs like GPT (Brown et al., 2020) and LLAMA (Touvron et al., 2023a;b) use Multi-Head Attention (MHA) (Vaswani et al., 2017). MHA uses several attention heads to look at a sequence. As models grow bigger, the number of heads increases as well. For example, LLAMA-7B uses 32 attention heads in each layer, while LLAMA-65B uses 64 attention heads per layer (Touvron et al., 2023a). The use of MHA exacerbates bottlenecks for serving LLMs. First, it increases compute pressure due to repeated application of the attention operation. Second, it increases the memory pressure due to requiring storage of Key (K), Value (V) caches that comes with the additional attention heads. To alleviate these bottlenecks, prior works have introduced primarily two types of methods - (i) pruning of LLMs to utilize sparsity based on the input context (Liu et al., 2023b; Voita et al., 2019) and (ii) Co-designing of the Attention module to reuse components across multiple heads like Multi-Query Attention (MQA) (Shazeer, 2019) and Grouped-Query Attention (GQA) (Ainslie et al., 2023).

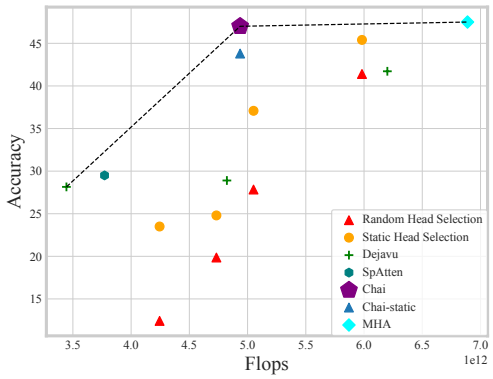


Figure 1. **Accuracy vs Flops:** We study various methods of clustering attention heads, and plot the runtime for sequence length of 2048. For random head selection we randomly choose heads to combine in increasing number of 4, 8, 16 and 24. For *Static Head Selection*, we choose the heads to combine based on activations. CHAI is our proposed method. These experiments are performed on LLAMA-7B on Arc-Challenge dataset.

[1]University of Wisconsin-Madison [2]Meta-FAIR. Correspondence to: Saurabh Agarwal <agarwal@cs.wisc.edu>. This work is carried out during Saurabh Agarwal's FAIR internship at Meta.

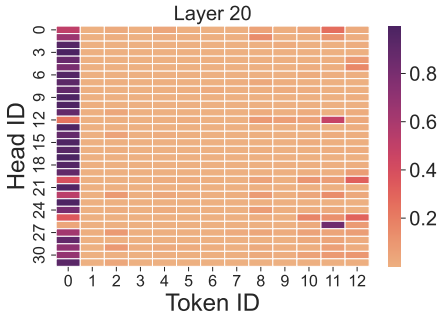
Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

Pruning LLMs can potentially ease the compute bottleneck. However, it is challenging as the classical methods for pruning (Frankle & Carbin, 2018; Chen et al., 2020b; You et al., 2019; Waleffe & Rekatsinas, 2020) require fine-tuning or iterative training which is prohibitively expensive for LLMs due to massive memory and compute cost. There have been recent pruning works such as DEJAVU (Liu et al., 2023b) which perform pruning based on the context at inference time without requiring fine-tuning. However, we observe that methods like DEJAVU are primarily designed for large parameter-inefficient models such as OPT (Zhang et al., 2022) and the insights used to build DEJAVU are not directly applicable on newer parameter efficient models like LLAMA-7B (Section 2). In Figure 1, we show that CHAI achieves the best trade-off between flops and accuracy compared to the state-of-the-art methods. Furthermore, runtime pruning methods like DEJAVU only reduce the compute cost and have no effect on the large memory requirements of K,V cache.

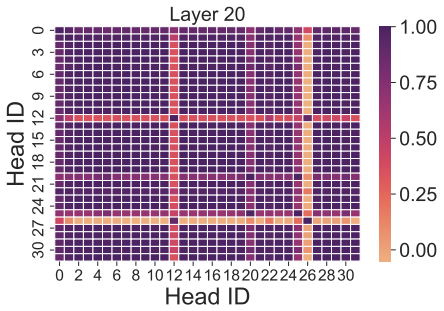
The Attention module co-design methods like GQA (Ainslie et al., 2023) require re-training of LLMs, *e.g.*, LLAMA-2 (Touvron et al., 2023b) trained the models from scratch to utilize the benefits of GQA, making it quite expensive. Even in the case where users are willing to perform retraining, accuracy trade-off between GQA and MHA will not be known prior to multiple rounds of training. Further, Attention module co-design methods only reduce the K,V cache size and do not reduce computational complexity. Therefore, there is a need for a method, which can reduce both the compute and K,V cache overhead for attention and is - (i) Applicable on a wide range of models (from LLAMA–7B to OPT-66B). (ii) Does not require any fine-tuning or re-training.

In this work we present Clustered Head Attention for efficient LLM Inference (CHAI), a dynamic inference time method for efficient LLM inference that does not require fine-tuning. CHAI is inspired by two observations. First, several heads in multi-head attention give similar weight to each token in a given sequence, indicating redundant compute. In Figure 2a we show attention scores for a single layer of LLAMA-7B for an auto-regressive decoding step of a sentence. We observe that several heads output similar scores, *i.e.*, giving similar weight to each token in the sequence. Figure 2b highlights the similarity in attention score by plotting correlation for the activation for LLAMA-7B. In Figure 2b we observe that there are three clusters and within these clusters the correlation is greater than 0.95.

This indicates that by identifying attention heads with similar attention scores and clustering them together we can reduce the number of self-attention operations for MHA by calculating self-attention only for a single head within a cluster. Secondly, we observe that for each request to an LLM we can accurately determine the heads which are going to give similar (attention) weight to the tokens in a sequence after running a few decoding steps on the sequence (Section 3.3). Schematic in Figure 3 depicts both Multi-Head and Clustered-Head Attention.



(a) **Activations of Multi Head Attention:** Figure shows activation scores for each token for each head. We observe that several heads give similar scores to the sequence.

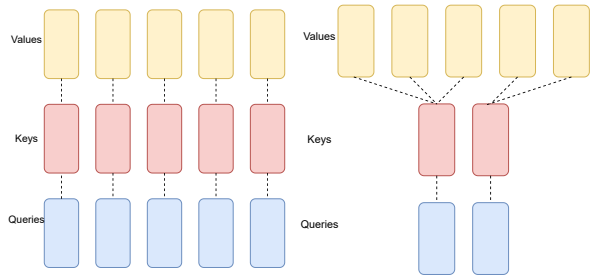


(b) **Pairwise cross correlation:** Pairwise cross-correlations show existence of three clusters- Heads [12,26] show strong correlation forming one cluster, Heads [20,25] form another, and the remaining heads form a third cluster.

Figure 2. **Redundancy across heads for LLAMA–7B**

Our contributions in this paper are as follows:

- We show that there is high level of redundancy across several different heads of multi head attention, and the redundancy varies differently across layers with increasing redundancy towards later layers.
- We introduce CHAI, a practical and principled inference time pruning method which clusters attention heads that have similar output together with dynamic determination of clusters. CHAI reduces both compute and K,V cache size for self attention.
- We show that CHAI is capable of reducing the inference time by up to 1.73× and K,V cache memory size by up to 21.4% compared to MHA for LLAMA models with minimal accuracy trade-off (maximum of 3.2%).
- Compared to other runtime pruning methods like DEJAVU, which only works well for OPT models, CHAI outperforms DEJAVU and performs well for wider class of models.



(a) **Multi-Head Attention** (b) **Clustered Head Attention**

Figure 3. **Clustered Head Attention:** Schematic of clustered head attention, comparing it with popular Multi-Head Attention. In clustered head attention, we remove the query and key vectors which produce similar attention scores.

2. Background and Related Work

We first provide background on inference process for decoder only transformers like GPT (Radford et al., 2019; Brown et al., 2020), LLaMa (Touvron et al., 2023a;b) and the bottlenecks in performing inference. Further, we discussed several prior lines of work which have tried to tackle the inference bottlenecks for transformer based model.

Decoder-only Transformer A decoder-only transformer forms the building block of popular LLMs. A single decoder block consists of a self attention layer and a MLP. An input token is fed into the decoder block, to perform next-word prediction. The self attention block uses prior query (Q), key (K) and value (V) vectors associated with current token. These tokens are extracted by performing a linear projection with query, key and value weight matrices associated with a transformer.

To precisely define Multi-Head Attention (MHA), let H, T, d be positive integers, where H denotes number of heads, T denotes sequence length, d denotes model dimension. Let $x \in \mathbb{R}^{T \times d}$ be input to the MHA layer. For a single head h, then $\mathbf{K}^h = x\mathbf{W}_K^h$, $\mathbf{Q}^h = x\mathbf{W}_Q^h$ and $\mathbf{V}^h = x\mathbf{W}_V^h$ denote the corresponding key, query and value vector. The attention matrix for head h is calculated as follows:

$$A_h = \sigma(\frac{1}{\sqrt{d}} Q^h K^{hT})$$

Output of MHA is denoted by:

$$y = A_0 V_0 \oplus A_1 V_1 \oplus A_2 V_2 \oplus \cdots \oplus A_H V_H$$

For performing inference, self attention needs access to the query, key and values associated with prior tokens. In order to avoid re-computation, inference serving systems cache the prior tokens in a sequence.

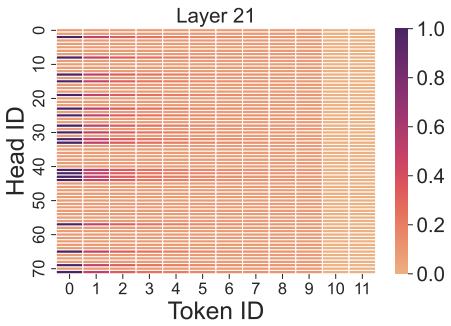
Compute cost required for multiple attention heads and memory capacity required for storing key and value vectors associated with each head during inference form two primary bottlenecks for LLM inference. In this work, we focus on reducing both memory and compute requirements via clustering multiple attention heads with similar output.

Building Efficient Transformers. Improving efficiency of transformer models has been of major focus in recent years. Prior work can be broadly categorized in the following fields - (i) Hardware-software co-design (Dao et al., 2022; Dao, 2023; Ham et al., 2020; 2021; Tambe et al., 2021; Fang et al., 2022; Qin et al., 2023; Wang et al., 2021b), (ii) Knowledge distillation (Hsieh et al., 2023; Jiao et al., 2019; Sanh et al., 2019; Wang et al., 2020) (iii) Neural Architecture Search (NAS) (Zhou et al., 2023; Kitaev et al., 2020; Lagunas et al., 2021) and (iv) Pruning (Voita et al., 2019; Liu et al., 2023b) and Quantization (Frantar et al., 2022; Xiao et al., 2023; Kim et al., 2021; Shen et al., 2020; Dettmers et al., 2022; Dettmers, 2015; Dettmers & Zettlemoyer, 2023). In this work our focus is on pruning , which we discuss next.

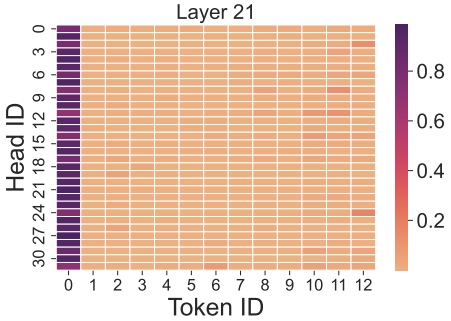
LLM Quantization. Recently several methods have been proposed to perform post training quantization allowing models to be quantized to a lower precision (Frantar et al., 2022; Xiao et al., 2023; Dettmers & Zettlemoyer, 2023). The goal of these methods is to perform quantization so as to minimize the error, CHAI is orthogonal to quantization based mechanisms as it depends on the insight of several attention heads focusing on the same tokens. The goal of quantization methods is to keep the same properties of original models, therefore we believe CHAI can be used to further accelerate post training quantized neural networks.

LLM Pruning. Pruning is a widely studied method to improve inference time by removing unused weights post training. Several prior works have looked at pruning for language models (Chen et al., 2020b; Prasanna et al., 2020; Chen et al., 2020a). For example, oBERT is a second order method to reduce the number of weights (Kurtic et al., 2022). Although these approaches can compress a model, they rarely yield inference speedups due to lack of hardware support for sparse operations on modern GPUs. To overcome the challenges, low rank decomposition methods (Wang et al., 2023; 2021a; 2019), attention head pruning (Michel et al., 2019; Voita et al., 2019), layer dropping (Sajjad et al., 2023; Fan et al., 2019; Dai et al., 2023) were proposed. However, these methods are infeasible for LLMs due to the use of iterative gradient calculations or fine-tuning leading to high resource requirements.

To overcome these issues, a recently proposed method, DEJAVU (Liu et al., 2023b), identifies portions of the model which are unused for a given context. To reduce the overhead of self-attention, DEJAVU prunes attention heads which give *uniform weight across tokens*. We plot the activations for an exemplary sentence used by DEJAVU for both OPT-66B and LLAMA-7B in Figure 4. We observe that while there are heads which give uniform weight to each token in OPT-66B model, there are no such heads in



(a) **OPT-66B**: For several heads the activation scores are uniform, *i.e.*, the heads given close to equal importance to each input token.



(b) **LLAMA-7B:** Heads in LLAMA-7B specifically pay attention to a specific token. However, multiple heads are attending to same token, in this case the first token.

Figure 4. **Activations for OPT-66B and LLAMA-7B for an exemplary sentence:** We observe that OPT-66B has several heads which give uniform attention scores to tokens whereas LLAMA-7B does not. However, both models have redundancies across heads, *i.e.*, groups of heads are give similar attention to each token.

more parameter efficient models like LLAMA-7B, indicating that for smaller parameter efficient models like LLAMA DEJAVU might not be applicable. (Additional plots for different layers can be found in Appendix-A.) The primary difference between OPT and LLAMA activation patterns could be attributed to the fact that LLAMA models are trained significantly longer and with more data.

We observe that CHAI's insight about redundancy in the output of multiple heads in the attention holds across both OPT and LLAMA family of models. In our evaluation (Section 4), we perform quantitative comparison between CHAI and DEJAVU.

K,V Cache Compression. Prior works which have tried to reduce the K,V cache size (Liu et al., 2023a; Zhang et al., 2023) by storing the K,V cache values for the most recent important tokens. However, they can not directly improve the latency of generating the next token, as they still perform the full transformer compute before finally deciding which K,V pairs should be stored. On the other hand, CHAI reduces not just the K,V cache size, it is also able to reduce the latency of next word prediction.

Speculative Decoding. Speculative decoding (Leviathan et al., 2023; Yang et al., 2023; Xia et al., 2023) is a popular

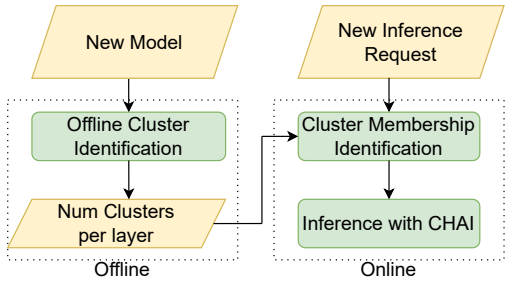


Figure 5. **CHAI Flow:** In the offline phase, we run clustering and perform elbow plot analysis for each new model. Then, for each new inference request we only perform cluster membership identification based on online performance.

method where a draft model is used to cheaply generate a sequence of draft tokens which can be efficiently verified by a target LLM. Speculative decoding can significantly reduce the latency of LLM serving, however it further exacerbates the compute and memory requirements as it requires additional resources to run both the draft and target model. CHAI on the other hand is focused on reducing the resource required for inference.

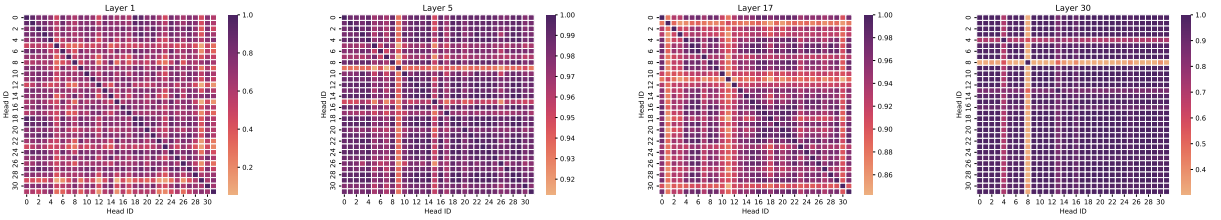
3. CHAI

Next, we describe CHAI. We first describe the key insights which have been used to build CHAI. Then, we detail CHAI's runtime pruning algorithm which is inspired by our insights and discuss how we perform inference using CHAI. Figure 5 provides a high level overview of inference using CHAI, which includes offline and online components.

3.1. Observations

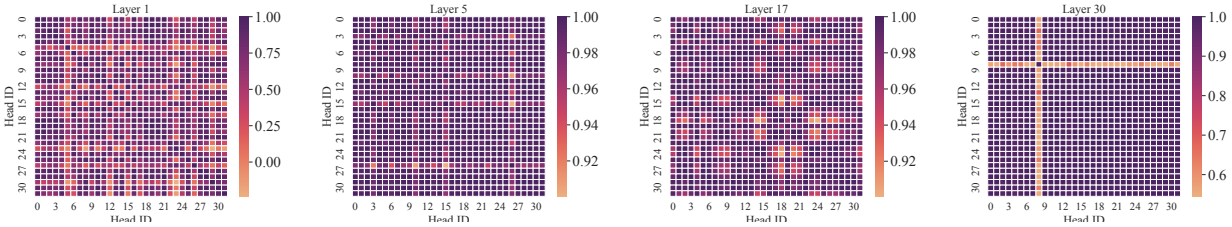
Our primary insight stems from the observation that there is a high amount of correlation across the output of various attention heads in MHA, *i.e.*, the output of several attention heads focuses on the same tokens. In Figure 6, we plot the average correlation across the 32 heads of LLAMA-7B for 1024 samples of the C4 (Raffel et al., 2020) dataset for different layers and in Figure 7, we plot correlation for a single sample of the dataset. These show us two insights - (i) Several heads output similar attention scores for each example and (ii) The amount of correlation increases in later layers, with heads in later layers with having higher correlation. This indicates that there is an opportunity to cluster attention heads with similar output and only run the self-attention operation for one of the representative attention heads within each cluster, thus reducing the amount of computation as well as the size of K,V cache.

Problem Formulation. Next, we formally define the problem of finding heads whose attention score is similar. Let H be the total number of attention heads, let $S = \{\langle K^1, Q^1\rangle, \langle K^2, Q^2\rangle, \langle K^3, Q^3\rangle, \cdots, \langle K^H, V^H\rangle\}$ be the set of Q, K pairs associated with each head h. Our goal



(a) Layer 1 (b) Layer 5 (c) Layer 17 (d) Layer 30

Figure 6. **Average Correlation for 1024 Samples of C4 on LLAMA-7B:** The above figure shows two interesting observations. First, there exists high amount of correlation across several heads of attention. Second, the correlation is not uniform across layers, with later layers having higher correlation, *i.e.*, , first layer has very little correlation but correlation increases in later layers.



(a) Layer 1 (b) Layer 5 (c) Layer 17 (d) Layer 30

Figure 7. **Correlation on a randomly selected single sample of LLAMA-7B.**

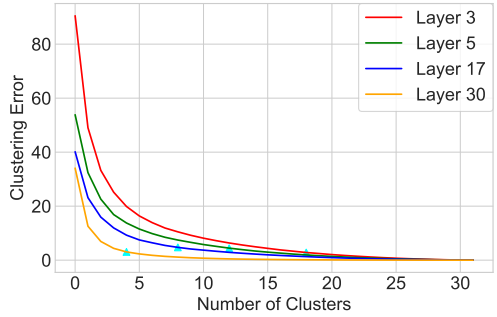


Figure 8. **Clustering Error:** We plot the clustering error on 1024 samples of C4-dataset. The markers represent the number of clusters we choose for a layer.

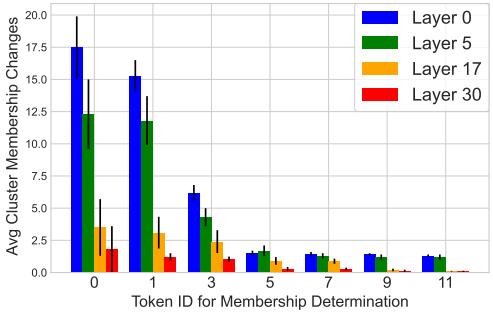


Figure 9. **Cluster Membership Evaluation:** We evaluate the number of times the cluster membership changes for performing next token prediction. We observed that if clustering is performed beyond the fifth token the number of times cluster membership changes is quite small.

is to find k subsets, $S_1 \subset S, S_2 \subset S, S_3 \subset S, \cdots S_k \subset S$ such that $< Q, K >$ pairs in each subset S_i produce similar output under function f. Where function f is the self attention operation, where $f(Q, K) = \sigma(QK^T)$. Further, we want $\cup_{i=1}^{k} S_i = S$.

Formally, we want to find S_i,

$$\forall < K^n, Q^n >, < K^m, Q^m > \in S_i,$$

s.t.

$$f(K^n, Q^n) \approx f(K^m, Q^m)$$

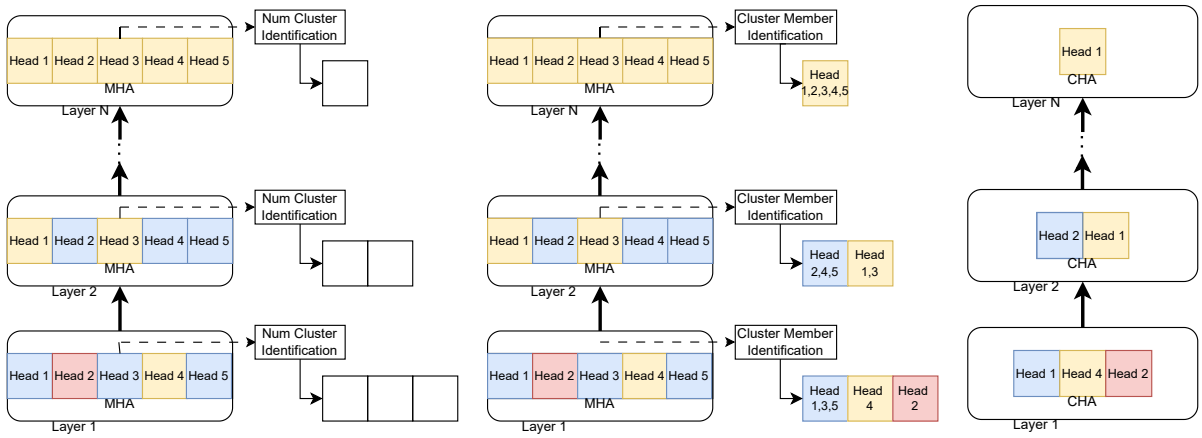
Informally, we want subset of heads, where within each subset the self attention operation gives similar outcome.

In order to solve this problem we need to determine k which represents the number of such subsets, and the membership of such subset S_i. Our observations empirically demonstrate the existence of such a solution. We can potentially solve this problem using clustering, where determining the number of subsets translates to determining number of clusters and determining cluster membership becomes determination of cluster membership.

To observe memory and compute savings, we need an accurate and efficient method to determine the number of clusters and their membership *without having access to activations*. Solving this forms a core contribution of our work.

3.2. Determination of Number of Clusters

Challenges. Figure 6 and Figure 7 indicate that the number of clusters varies widely per layer in a LLM. Specifically, the last few layers in the LLM exhibit a very low number of clusters (high redundancy), whereas the early layers demonstrate a high degree of variance across the output of heads resulting in large number of clusters. This observation suggests that the method used to determine number of clusters needs to make decisions for each layer



(a) **Offline Cluster Identification:** For each new model we run an offline cluster identification phase. We collect the activations and perform Elbow-plot analysis to decide number of clusters.

(b) **Cluster Membership Identification:** For each new request, we initial run with multi-head attention for first five tokens. Using this we determine the number of clusters in each layer.

(c) **CHAI Inference:** Post cluster membership identification we substitute MHA with Clustered Head Attention.

Figure 10. **Schematic of CHAI detailing three phases of the system.**

independently. Additionally, widely used methods such as Elbow plot method (Thorndike, 1953) for determining number of clusters entail manual effort making cluster number determination impractical at inference time.

Design. To determine the number of clusters, we propose an offline strategy we run once for each model. In our case, we sample a small number of samples (1024) from the C4 (Raffel et al., 2020) dataset and perform elbow-plot analysis by plotting clustering error (i.e. sum of squared distance from the closest cluster) as a function of number of clusters. Figure 8 shows the clustering error for LLAMA-7B for the samples selected. Based on the Elbow-plot analysis we choose the number of clusters when the error plateaus.

The offline analysis is performed once for each network by using the C4 (Raffel et al., 2020) dataset. We do not change the number of clusters determined for a new dataset.

3.3. Determination of Cluster Membership

Challenges. Having determined number of clusters, we need to determine the membership of these clusters, *i.e.*, which heads belong to which cluster in each layer. For Figure 6, 7 and 8, we perform clustering based on activations obtained by performing the forward pass. However, for each decoding step, performing clustering on output of self attention post forward pass will not yield any performance benefit as we will still be performing the original compute and using the full K,V cache. In order to utilize the insights observed in Section 3.1, we will need to decide the cluster members without having access to the output of the self attention.

Design. A simple strategy would have been keeping the cluster membership static across the tokens and independent of input context, *e.g.*, we use the same cluster membership found during offline analysis with C4 data in the previous section. For evaluation purposes, we call this version of head selection **CHAI-static**.

However, we observed that the cluster membership does not remain static and varies based on context. When comparing Figure 7, which plots correlation for a single example, with Figure 6, which plots correlation for 1024 samples, we observe that the correlation across heads varies with varying context. Therefore, the correlation across the output of the heads depends on the context (input prompt), *i.e.*, *a solution to determine the membership of each cluster has to account for context.* To understand the effects of accounting for context while clustering heads, we analysed the change in cluster membership changes and clustering with different context. In Figure 9, we observed an interesting phenomenon, after determining cluster membership by accounting for five tokens, the cluster membership does not change frequently. A direct outcome of this observation is that for each new sequence we can perform clustering based on the output of self-attention after the first five tokens. We observe that *activation from first five tokens of a new sequence are enough to accurately predict the cluster membership.* This dynamic version of head selection further allows us to improve accuracy over CHAI-static. Figure 10b shows an illustration of the membership identification step. Furthermore, evaluation results in Section 4 compare CHAI-static and CHAI performance.

Table 1. Accuracy on OPT-66B

Method	PIQA	Hellaswag	Arc-Challenge	Arc-Easy	Boolq
MHA	78.4	71.1	41.6	64.7	65.4
DejaVu-50%	-0.25	-0.7	-0.6	-0.2	-4.0
CHAI-static	-1.35	-1.7	-0.7	-0.7	-0.7
CHAI	**-0.15**	**0.1**	**0.1**	**-0.1**	**-0.6**

Table 2. Accuracy on LLAMA-7B

Method	PIQA	HellaSwag	Arc-Challenge	Arc-Easy	BoolQ
MHA	79.8	76.1	47.5	72.8	76.0
DejaVu-10%	-3.9	-4.7	-5.78	-3.18	-7.4
DejaVu-30%	-13.3	-18.6	-18.75	-4.2	-20.2
DejaVu-50%	-24.6	-50.7	-19.35	-46.3	-21.6
SpAtten	-41.4	-42.5	-18.0	-40.2	-27.1
CHAI-static	-4.0	-4.3	-3.7	-2.5	-0.8
CHAI	**-2.0**	**-3.2**	**-0.5**	**0.3**	**0.1**

Table 3. Accuracy on LLAMA-33B

Method	PIQA	HellaSwag	Arc-Challenge	Arc-Easy	BoolQ
MHA	82.1	82.8	57.8	80.0	83.1
DejaVu-10%	-0.7	0.1	**-0.2**	-0.6	-0.2
DejaVu-30%	-9.3	-24.4	-17.91	-12.4	-12.2
DejaVu-50%	-27.6	-43.2	-24.6	-37.6	-21.2
SpAtten	-31.9	-44.1	-26.4	-40.3	-34.55
CHAI-static	-0.5	-0.2	-1.3	-3.7	-1.5
CHAI	**0**	**-0.14**	-0.21	**0.9**	**-0.04**

3.4. Clustered Head Attention

Once we have decided which heads have similar attention output, we can than use Clustered Head Attention to combine key and query vectors for the heads.

3.5. Inference using CHAI

Next we, discuss the inference flow of CHAI, illustrated in detail in Figure 10. For each new model we first perform offline cluster identification (Figure 10a). Then for each new request, we determine the cluster membership using K-Means clustering once we have processed five tokens, using the observed activations (Figure 10b). After this step, we keep the clustered heads same throughout inference (Figure 10c).

There are two direct outcomes of CHAI's design. First, we directly reduce the amount of computation by removing redundant heads. Secondly, after a pre-determined token we fix the heads which are going to be pruned, this also allows us to remove the corresponding *Key* tokens associated, which significantly reduces the K,V cache size. Therefore, CHAI allows us to reduce both the inference compute as well as the size of the K,V cache required.

4. Evaluation

We experimentally verify the performance of CHAI and compare it to DEJAVU (Liu et al., 2023b) and SPATTEN (Wang et al., 2021b) on three different models of various sizes LLaMa-7B (Touvron et al., 2023a), LLaMa-33B and OPT-66B (Zhang et al., 2022). We evaluate the models on five commonly used NLP tasks: PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), Arc-Challenge and Arc-Easy (Clark et al., 2018) and BoolQA (Clark et al., 2019).

4.1. Experimental Setup

All our experiments are performed on servers with NVIDIA V100 GPUs. For OPT-66B we used eight GPUs on a single node, for LLAMA-33B we used four GPUs, and for LLAMA-7B, we used a single GPU for inference. CHAI is built on top of Meta's xFormers (facebookresearch, 2023).

4.2. Accuracy Evaluation

In our evaluation, we compare CHAI with Multi-Head Attention as baseline, static version of CHAI, as well two other state-of-the-art prior pruning methods; DEJAVU and SPATTEN. For DEJAVU, we try different sparsity ratios, in order to try to match the accuracy number to MHA. We also compare CHAI to SPATTEN, a method which removes unimportant tokens and heads.

In Table 1, we first verify that we are able to reproduce the performance numbers reported by DEJAVU. To perform this, we took the OPT-66B and evaluated both DEJAVU, CHAI and CHAI-static. We used DEJAVU with 50% sparsity as reported by the authors. We used the author provided code to train their MLP predictor layers and incorporate their scheme in our setup. In Table 1, we observe that we were able to replicate results for OPT-66B. Furthermore, CHAI is also able to match the accuracy of MHA for OPT-66B.

Next, we compare CHAI, CHAI-static and DEJAVU with the pre-trained MHA network, using LLAMA-7B on 5 different datasets. For DEJAVU we used three configurations, 50% sparsity, 30% sparsity and 10% sparsity. In Table 2, we observe that when we use DEJAVU with more 10% sparsity we see significant decrease in accuracy (by 18.6% for DEJAVU-30%). On the other hand, our method based on our close analysis of the behaviour of layers of LLAMA-7B is able to recover accuracy. We observe a maximum accuracy degradation of 3.7% for CHAI. Similarly for LLAMA-33B using sparsity for more than 10% leads to significant accuracy drop, meanwhile CHAI closely matches the accuracy of the pre-trained model using MHA with maximum degradation in accuracy by 0.14%. This shows that CHAI is widely applicable across multiple datasets and models. We also want to highlight that we do not perform any dataset specific tuning.

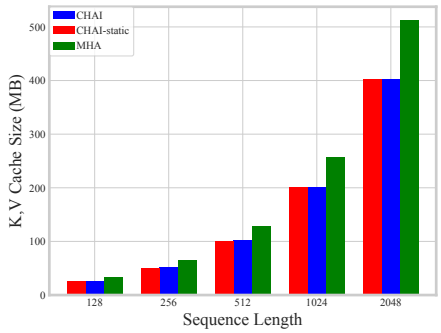


Figure 11. **Memory Savings**: We observed that for LLAMA-7B CHAI provides memory savings of up to 21.4%.

4.3. Memory Capacity Evaluation

CHAI reduces memory capacity requirements due to reduction in K,V cache size and minimal additional storage required to store the cluster map. In Figure 11, we show that for LLAMA-7B CHAI reduces the size of K,V cache by up to 21.4% compared to MHA. Even for comparatively small models like LLAMA-7B, the size of the K,V cache for a sequence length of 2048 is around 1.2 GB, while around 12 GB is used for the model weights. A reduction in K,V cache size can enable use of larger context length or serving more requests. We would also like to note that as shown in Figure 3, CHAI only removes the keys associated with redundant heads and keeps all the value vectors.

Memory overhead of CHAI. The only additional storage CHAI requires is storing the cluster map. Size of the map can be determined by $n_layers \times (n_heads + n_clusters)$, for LLama7B the number of layers is 32, and number of heads is 32 and number of clusters vary from 28 in early layers to 4 in most of the later layers. Thus storing this map only requires a few hundreds of bytes. We would like to point out that similar methods like DejaVu require training MLP classifiers per layer to learn the sparsity pattern which need to be stored.

4.4. End-to-End Latency Evaluation

Next, we evaluate time to first token and time to next token comparing it with MHA. These are two standard metrics used for evaluation of an LLM. Time to first token evaluates the time for generating a first token given a new context. Time to first token accounts for generating K,V caches for all the tokens in the context. Whereas time to next token evaluates the time for generating the next token, assuming the K,V caches for all internal tokens is available.

Time to first token. Next, in our experiments we compare the speedups provided by CHAI. In Figure 12-(a) for LLAMA-7B we show that our method provides speedup of up to 1.72× on a sequence length of 2048. The execution times represented in this figure accounts for the overhead of clustering in CHAI.

Time to next token. Another metric for evaluation of LLMs is time to next token. We do not account for the overhead of clustering in the case of time to next token. Our primary wins come from reducing compute and reducing memory bandwidth requirement for performing time to next token. Figure 12-(b) shows time to predict the next token for different sequence lengths. We observe that CHAI provides a speedup of over 5× for a sequence length of 2048.

Unfortunately, we are not able to compare times with DEJAVU as the authors have not released the specialized kernels used for realizing the speedups on hardware (git, 2024), thus inhibiting a runtime comparison. However, we believe it is unlikely that at less than 10% sparsity which is needed by DEJAVU to get comparable accuracy to MHA, it will yield high speedups (Hooker, 2021). We would like to highlight that because of performing dense computations, unlike DEJAVU, CHAI does not need custom GPU kernels. Further, CHAI's speedup benefits are independent of the framework used, because irrespective of implementation, CHAI directly reduces the complexity of MHA.

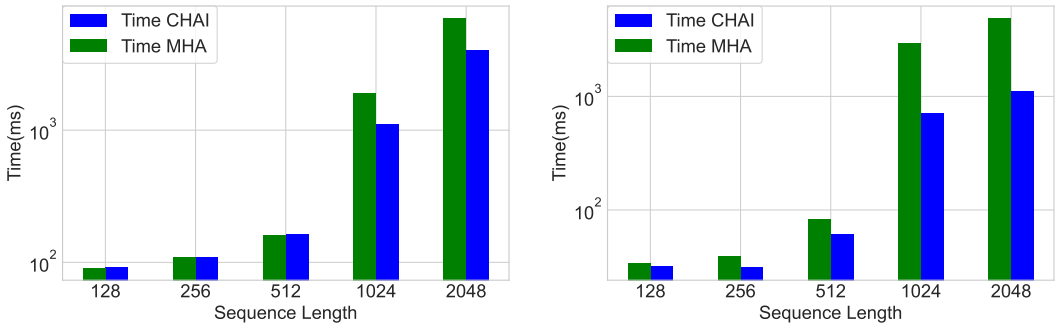
Compute overhead of CHAI. Existing efficiency enhancing methods require some fine-tuning or modification to the architecture. While DejaVu (Liu et al., 2023b) and SpAtten (Wang et al., 2021b) are runtime methods they still require additional compute, *e.g.*, DejaVu requires running MLP classifiers during inference. CHAI on the other hand only adds the computation overhead for determining the cluster membership. In our experiments we observed that clustering takes only 0.6 ms per request, which is about 0.008% of the inference latency.

4.5. Additional Experiments

Next we perform additional studies on our algorithm.

CHAI with Quantization. Next, we run experiments to understand how CHAI performs in conjunction with quantization. To perform this experiment we take an open source GPTQ (Frantar et al., 2022) quantized model from HuggingFace (TheBloke, 2023) and run CHAI on the model. In Table 4, we show the performance of CHAI on a LLAMA-7B on the 4-bit quantized model. We observe that the maximum deviation from accuracy for the quantized model is 0.1%.

CHAI with Grouped Query Attention. Grouped Query Attention (Ainslie et al., 2023) is a widely used method which shares single Key and Value vectors across multiple queries. This effectively reduces the K,V cache size. We perform preliminary studies to understand how CHAI can be used in conjunction with GQA. For these experiments we used models like LLama 2 - 70B that are pre-trained with GQA, we ran some preliminary experiments where we



(a) **Time to first token:** We observe speedups of up to 1.73× for sequence length of 2048.

(b) **Time to next token:** We observe a speedup of up to 5× for sequence length of 2048.

Figure 12. **Latency Analysis:** We observe that the speedups provided by CHAI increases as the sequence length becomes larger. Even for a comparatively small model like LLAMA-7B we observe speedups of up to 1.73× for a large sequence length.

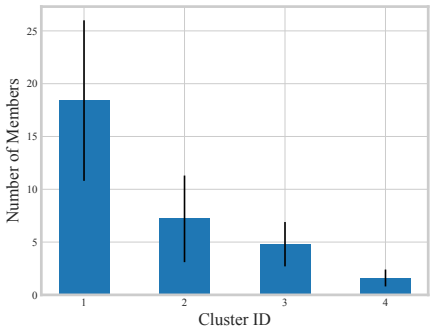


Figure 13. **Cluster Distribution**: We observe that number of heads within the cluster is quite skewed. We often observe one or two large clusters, while the remaining heads in the cluster.

Table 4. CHAI with Quantization

	HellaSwag	PIQA	BoolQ	Arc-Challenge	Arc-Easy
LLama-7B	76.1	79.8	76.0	47.5	72.8
LLama-7B-4bit-GPTQ	67.97	71.34	66.73	41.23	64.8
CHAI-LLama-7B-4bit-GPTQ	67.91	72.05	66.48	41.19	64.7

used CHAI to further reduce the number of K,Q pairs in grouped query attention. LLama-70B uses grouping factor of 8, i.e., 8 queries map to single key vector, we were able to reduce the number of K vectors on average by 1.8x and Q vectors by 3.7x. In Table 5. We show this reduction leads to negligible degradation in accuracy. This indicates that it is possible to use CHAI with GQA.

Pruning K, Q and V. In CHAI, we prune only the Key and Query portion of an attention head leaving the Value vector intact. Next, we study how accuracy changes if we remove the value vector as well. To perform this experiment we chose to reuse the value vector generated by the chosen head. In Table 6, we show how reusing the full head (Query, Key and Value vector) lead to additional loss in accuracy. This shows that for smaller networks like LLAMA it might be hard to remove the whole head in Multi-Head Attention.

Cluster Distribution. Figure 13 shows the distribution across clusters for Layer-18 on LLAMA-7B for different

Table 5. CHAI with GQA

	Hellaswag	PIQA	BoolQ
LLama2-70B-GQA	85.3	82.8	85.0
CHAI-LLama2-70B-GQA	85.2	82.73	85.0

Table 6. Pruning Both Q,K,V

	CHAI	CHAI-QKV	MHA
Arc-Challenge	47.0	41.29	47.5
PIQA	77.8	61.93	79.8

1024 samples of C4 dataset. We observe that typically for LLMs majority of heads can be grouped into a single head.

5. Conclusion

In this work, we present CHAI, an efficient runtime method which identifies attention heads giving similar scores. Using this method we reduce overhead of Multi-Head Attention by clustering the correlated heads and computing attention scores only for heads which lead to disparate attention scores. Our evaluation shows that with minor accuracy loss system can speedup inference by up to 1.73×.

Impact Statement

Our work aims to improve the efficiency of LLM inference and would boost the latency and throughput of LLM inference. We aim to contribute towards reducing the cost of LLM inference, making a positive environmental impact by using fewer computational resources. Our work would not maliciously alter an LLM output, and therefore would not generate any additional harm an LLM may cause that we feel must be specifically highlighted here.

References

Deja Vu: Contextual Sparsity for Efficient LLMs

at Inference Time. https://github.com/FMInference/DejaVu, 2024.

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chen, T., Frankle, J., Chang, S., Liu, S., Zhang, Y., Wang, Z., and Carbin, M. The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems*, 33:15834–15846, 2020a.

Chen, X., Cheng, Y., Wang, S., Gan, Z., Wang, Z., and Liu, J. Earlybert: Efficient bert training via early-bird lottery tickets. *arXiv preprint arXiv:2101.00063*, 2020b.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Dai, S., Genc, H., Venkatesan, R., and Khailany, B. Efficient transformer inference with statically structured sparse attention. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2023.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Dettmers, T. 8-bit approximations for parallelism in deep learning. *arXiv preprint arXiv:1511.04561*, 2015.

Dettmers, T. and Zettlemoyer, L. The case for 4-bit precision: k-bit inference scaling laws. In *International Conference on Machine Learning*, pp. 7750–7774. PMLR, 2023.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

facebookresearch. xformers - toolbox to accelerate research on transformers. https://github.com/facebookresearch/xformers, 2023. Accessed: December 12, 2023.

Fan, A., Grave, E., and Joulin, A. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*, 2019.

Fang, C., Zhou, A., and Wang, Z. An algorithm–hardware co-optimized framework for accelerating n: M sparse transformers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30(11):1573–1586, 2022.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Ham, T. J., Jung, S. J., Kim, S., Oh, Y. H., Park, Y., Song, Y., Park, J.-H., Lee, S., Park, K., Lee, J. W., et al. Aˆ 3: Accelerating attention mechanisms in neural networks with approximation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 328–341. IEEE, 2020.

Ham, T. J., Lee, Y., Seo, S. H., Kim, S., Choi, H., Jung, S. J., and Lee, J. W. Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 692–705. IEEE, 2021.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Hooker, S. The hardware lottery. *Communications of the ACM*, 64(12):58–65, 2021.

Hsieh, C.-Y., Li, C.-L., Yeh, C.-K., Nakhost, H., Fujii, Y., Ratner, A., Krishna, R., Lee, C.-Y., and Pfister, T. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. *arXiv preprint arXiv:2305.02301*, 2023.

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., and Liu, Q. Tinybert: Distilling bert

for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kim, S., Gholami, A., Yao, Z., Mahoney, M. W., and Keutzer, K. I-bert: Integer-only bert quantization. In *International conference on machine learning*, pp. 5506–5518. PMLR, 2021.

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

Kurtic, E., Campos, D., Nguyen, T., Frantar, E., Kurtz, M., Fineran, B., Goin, M., and Alistarh, D. The optimal bert surgeon: Scalable and accurate second-order pruning for large language models. *arXiv preprint arXiv:2203.07259*, 2022.

Lagunas, F., Charlaix, E., Sanh, V., and Rush, A. M. Block pruning for faster transformers. *arXiv preprint arXiv:2109.04838*, 2021.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Liu, Z., Desai, A., Liao, F., Wang, W., Xie, V., Xu, Z., Kyrillidis, A., and Shrivastava, A. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *arXiv preprint arXiv:2305.17118*, 2023a.

Liu, Z., Wang, J., Dao, T., Zhou, T., Yuan, B., Song, Z., Shrivastava, A., Zhang, C., Tian, Y., Re, C., et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pp. 22137–22176. PMLR, 2023b.

Michel, P., Levy, O., and Neubig, G. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.

Prasanna, S., Rogers, A., and Rumshisky, A. When bert plays the lottery, all tickets are winning. *arXiv preprint arXiv:2005.00561*, 2020.

Qin, Y., Wang, Y., Deng, D., Zhao, Z., Yang, X., Liu, L., Wei, S., Hu, Y., and Yin, S. Fact: Ffn-attention co-optimized transformer architecture with eager correlation prediction. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pp. 1–14, 2023.

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

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

Sajjad, H., Dalvi, F., Durrani, N., and Nakov, P. On the effect of dropping layers of pre-trained transformer models. *Computer Speech & Language*, 77:101429, 2023.

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

Shazeer, N. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

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

Tambe, T., Hooper, C., Pentecost, L., Jia, T., Yang, E.-Y., Donato, M., Sanh, V., Whatmough, P., Rush, A. M., Brooks, D., et al. Edgebert: Sentence-level energy optimizations for latency-aware multi-task nlp inference. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 830–844, 2021.

TheBloke. Llama-7b gptq. https://huggingface.co/TheBloke/LLaMa-7B-GPTQ, 2023. Accessed: December 12, 2023.

Thorndike, R. L. Who belongs in the family? *Psychometrika*, 18(4):267–276, 1953.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Voita, E., Talbot, D., Moiseev, F., Sennrich, R., and Titov, I. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*, 2019.

Waleffe, R. and Rekatsinas, T. Principal component networks: Parameter reduction early in training. *arXiv preprint arXiv:2006.13347*, 2020.

Wang, H., Agarwal, S., and Papailiopoulos, D. Pufferfish: Communication-efficient models at no extra cost. *Proceedings of Machine Learning and Systems*, 3:365–386, 2021a.

Wang, H., Zhang, Z., and Han, S. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 97–110. IEEE, 2021b.

Wang, H., Agarwal, S., Tanaka, Y., Xing, E., Papailiopoulos, D., et al. Cuttlefish: Low-rank model training without all the tuning. *Proceedings of Machine Learning and Systems*, 5, 2023.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Wang, Z., Wohlwend, J., and Lei, T. Structured pruning of large language models. *arXiv preprint arXiv:1910.04732*, 2019.

Xia, H., Ge, T., Wang, P., Chen, S.-Q., Wei, F., and Sui, Z. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3909–3925, 2023.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pp. 38087–38099. PMLR, 2023.

Yang, S., Lee, G., Cho, J., Papailiopoulos, D., and Lee, K. Predictive pipelined decoding: A compute-latency trade-off for exact llm decoding. *arXiv preprint arXiv:2307.05908*, 2023.

You, H., Li, C., Xu, P., Fu, Y., Wang, Y., Chen, X., Baraniuk, R. G., Wang, Z., and Lin, Y. Drawing early-bird tickets: Towards more efficient training of deep networks. *arXiv preprint arXiv:1909.11957*, 2019.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H _2 o: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv:2306.14048*, 2023.

Zhou, Y., Du, N., Huang, Y., Peng, D., Lan, C., Huang, D., Shakeri, S., So, D., Dai, A. M., Lu, Y., et al. Brainformers: Trading simplicity for efficiency. In *International Conference on Machine Learning*, pp. 42531–42542. PMLR, 2023.

A. Additional Plots

A.1. Accuracy and Inference Time Trade-off

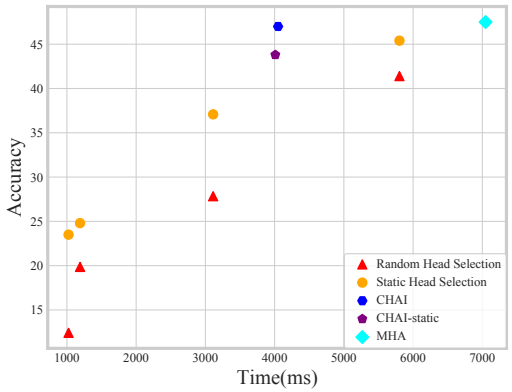


Figure 14. **Accuracy vs Inference Time for LLAMA-7B:** We study various methods of clustering attention heads, and plot the runtime for sequence length of 2048. For random head selection we randomly choose heads to combine together in increasing number of 4, 8, 16 and 24. For Static Head selection we choose the heads in increasing order of 4,8,16, and 24 based on activation analysis of activation on C4 dataset (Raffel et al., 2020).

A.2. Dataset Ablation to determine number of clusters

We perform additional experiments to understand affect of dataset to determine the number of clusters. In Figure 15 we show that number of clusters determined using different datasets is same. Thus indicating that number of clusters is independent of dataset.

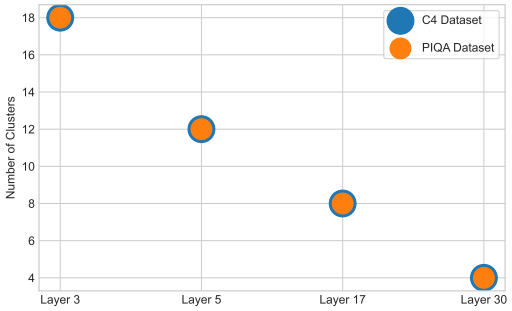


Figure 15. **Dataset Ablation to determine number of cluster:** We study how change in dataset affects the number of clusters determined for a network. For this analysis we plot the number of clusters determined by 1024 samples of PIQA (Bisk et al., 2020) and C4 (Raffel et al., 2020)

A.3. Sample Ablation to determine number of Clusters

Next we perform ablation to understand how choice of number of samples affect the number of clusters determined. In Figure 16 we show that as long as the sample size is more than 256 we observe similar number of clusters.

A.4. Latency Breakdown under CHAI

In Figure 17 we show the latency breakdown under CHAI.

A.5. OPT-66B Activation Plots

From Figure 18 to Figure 18 shows the activation plots for all layers of OPT. We consistently observe that for this examples there is high amount of.

A.6. LLAMA-7B Activation Plots

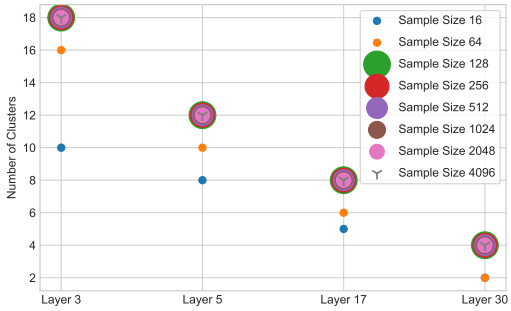


Figure 16. **Sample Ablation to determine number of cluster:** We study how change in number of samples affects the number of clusters determined for a network. For this analysis we plot the number of clusters determined with different number of samples. We observe that as long as the number of samples is more than 256 we observe similar number of clusters.

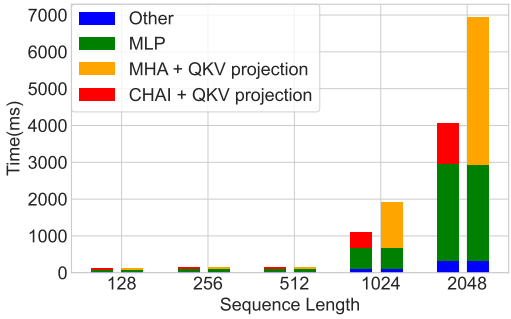
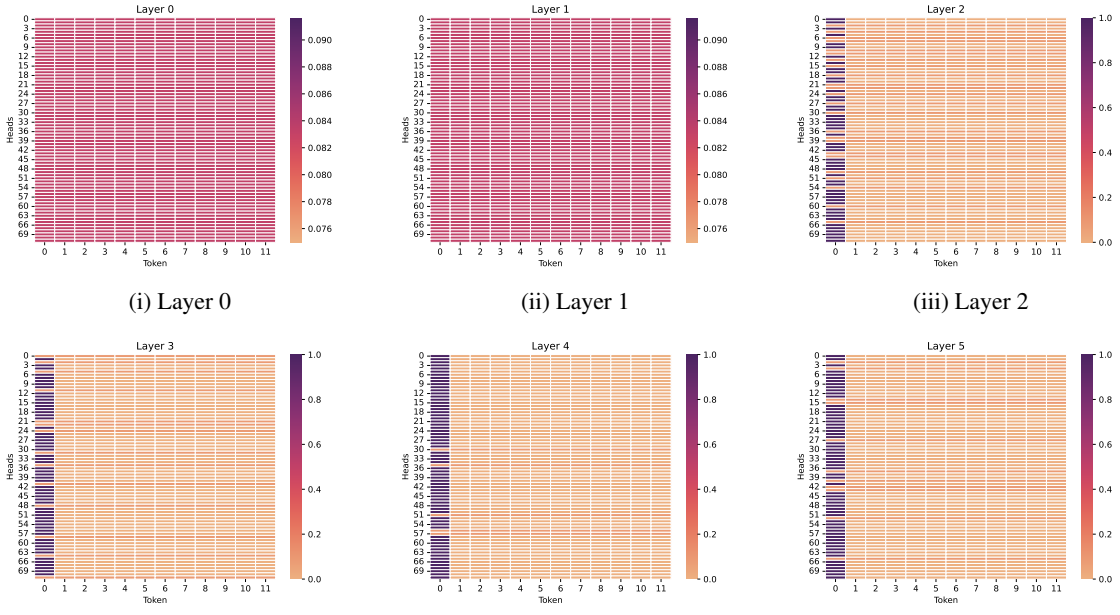


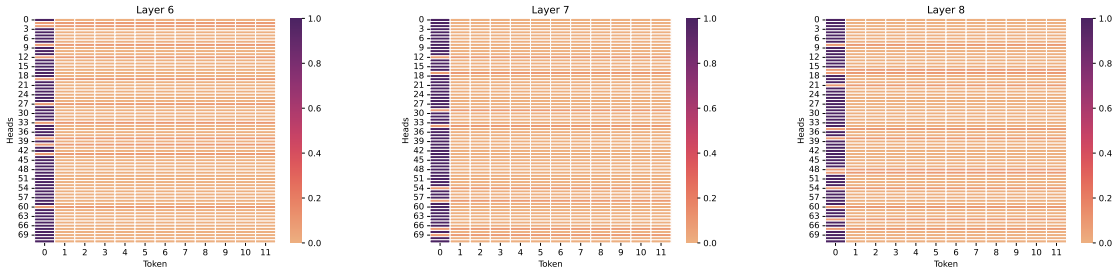
Figure 17. **Time breakdown for generating first token:** We plot the breakdown of time spent across different component of a LLM. We observe that CHAI primarily reduces only the projection and compute time.



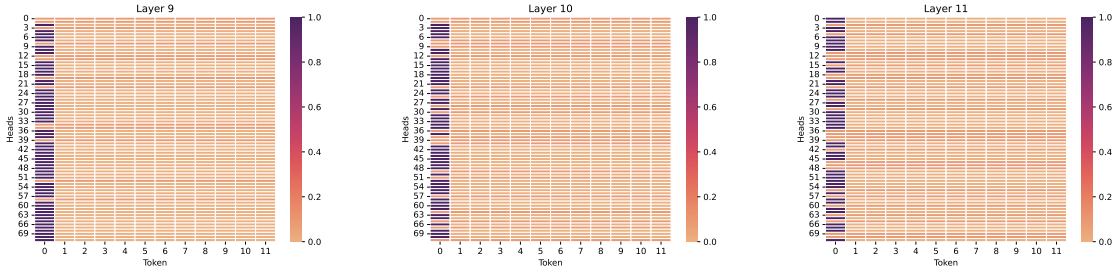
(i) Layer 0 (ii) Layer 1 (iii) Layer 2

(iv) Layer 3 (v) Layer 4 (vi) Layer 5

Figure 18. Activations for OPT-66B

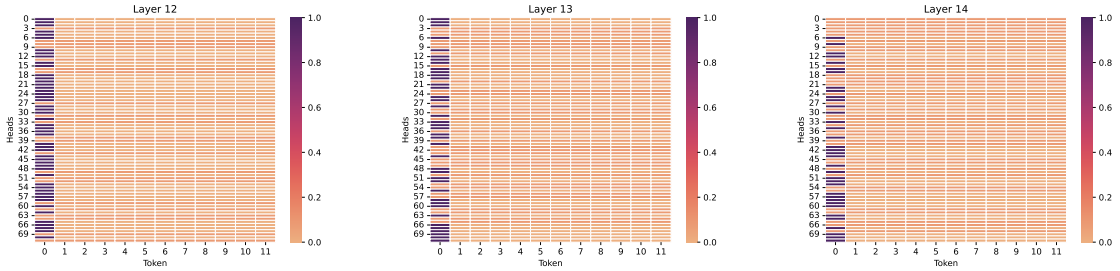


(vii) Layer 6 (viii) Layer 7 (ix) Layer 8

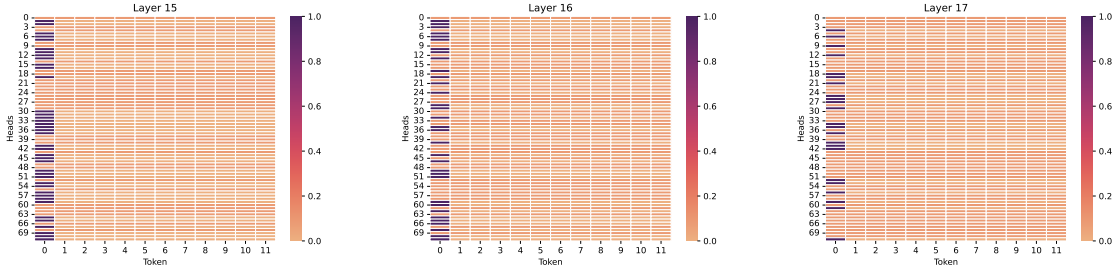


(x) Layer 9 (xi) Layer 10 (xii) Layer 11

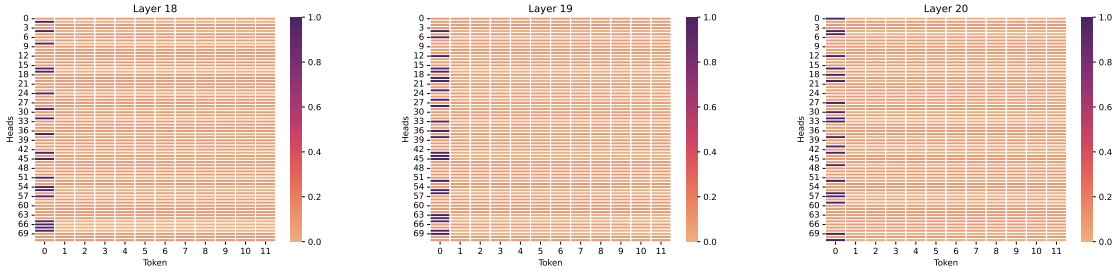
Figure 18. Activations for OPT-66B



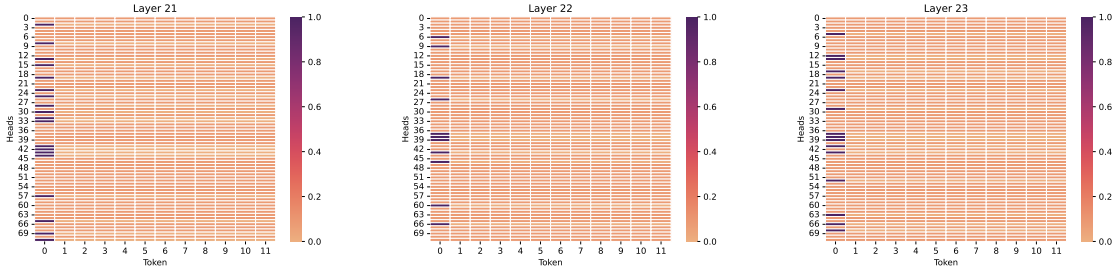
(xiii) Layer 12 (xiv) Layer 13 (xv) Layer 14



(xvi) Layer 15 (xvii) Layer 16 (xviii) Layer 17

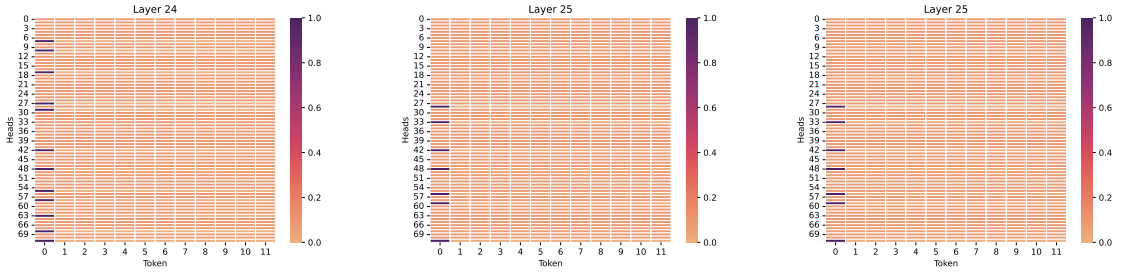


(xix) Layer 18 (xx) Layer 19 (xxi) Layer 20

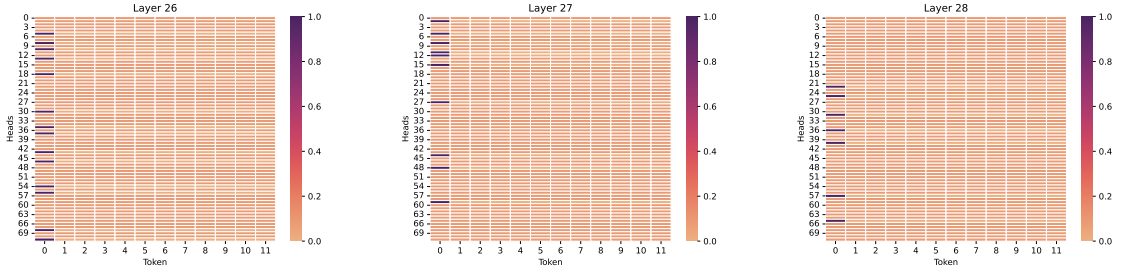


(xxii) Layer 21 (xxiii) Layer 22 (xxiv) Layer 20

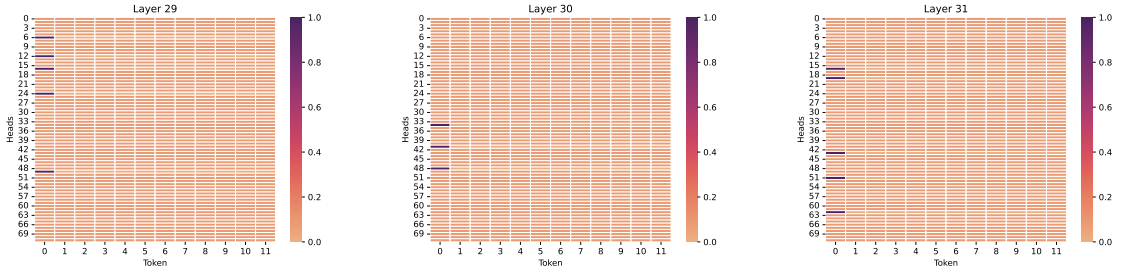
Figure 18. Activations of OPT-66B



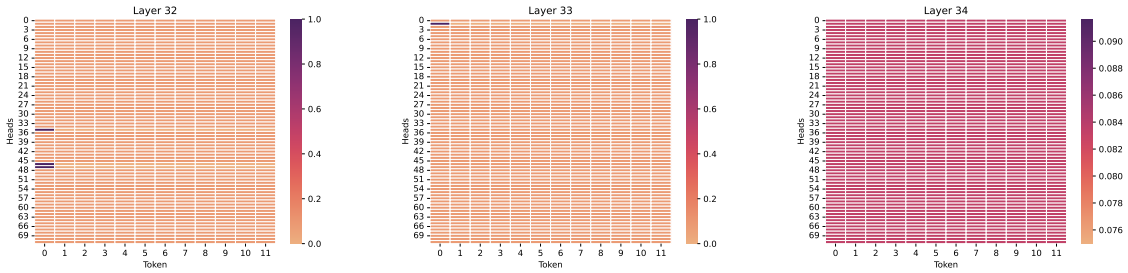
(xxv) Layer 24 (xxvi) Layer 25 (xxvii) Layer 25



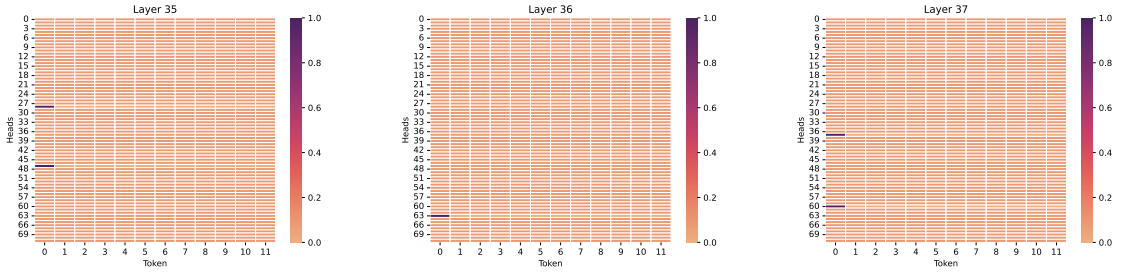
(xxviii) Layer 26 (xxix) Layer 27 (xxx) Layer 28



(xxxi) Layer 29 (xxxii) Layer 30 (xxxiii) Layer 31

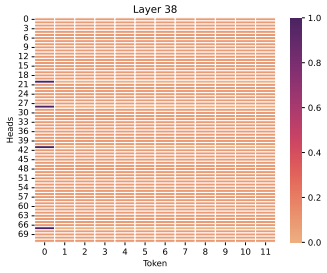


(xxxiv) Layer 32 (xxxv) Layer 33 (xxxvi) Layer 34

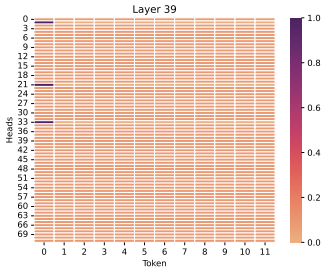


(xxxvii) Layer 35 (xxxviii) Layer 36 (xxxix) Layer 37

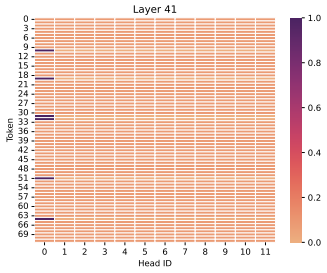
Figure 18. Activations of OPT-66B



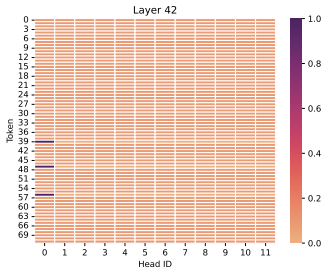
(xl) Layer 38



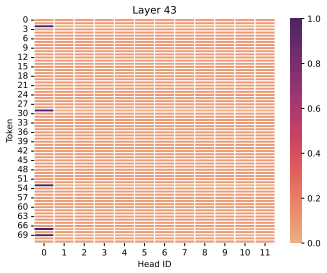
(xli) Layer 39



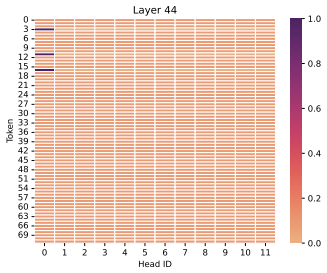
(xlii) Layer 41



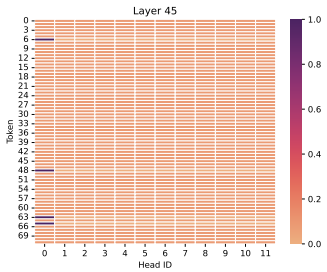
(xliii) Layer 42



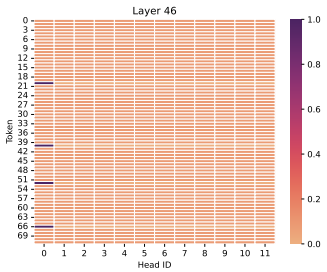
(xliv) Layer 43



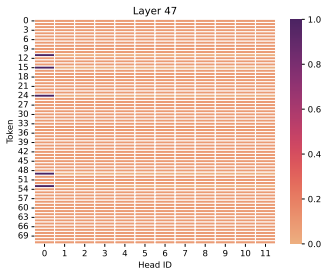
(xlv) Layer 44



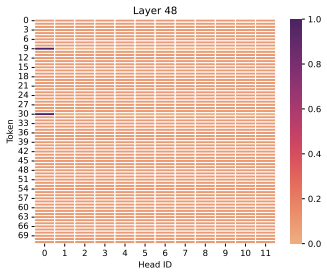
(xlvi) Layer 45



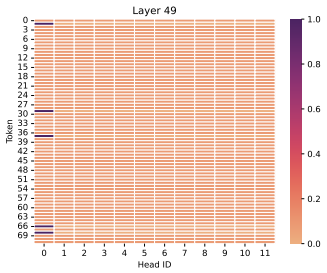
(xlvii) Layer 46



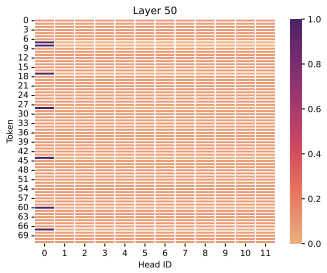
(xlviii) Layer 47



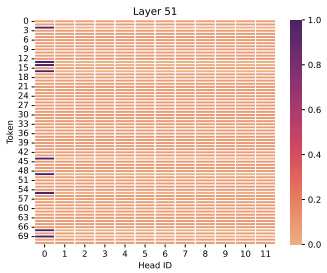
(xlix) Layer 48



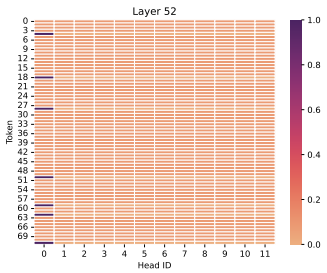
(l) Layer 49



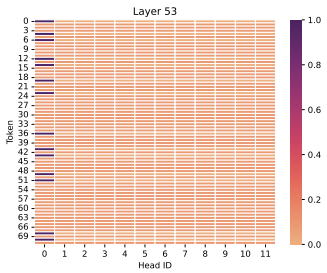
(li) Layer 50



(lii) Layer 51

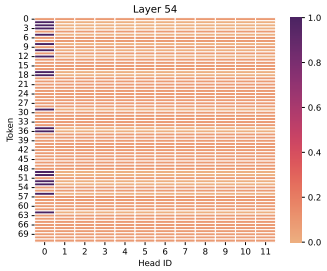


(liii) Layer 52

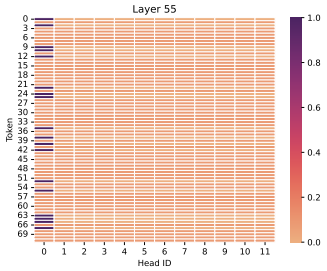


(liv) Layer 53

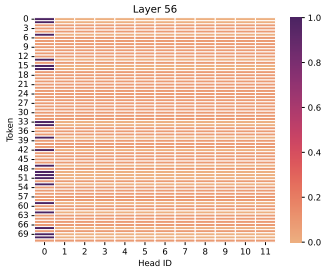
Figure 18. Activations of OPT-66B



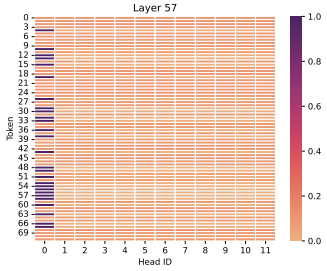
(lv) Layer 54



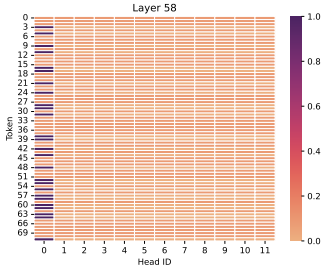
(lvi) Layer 55



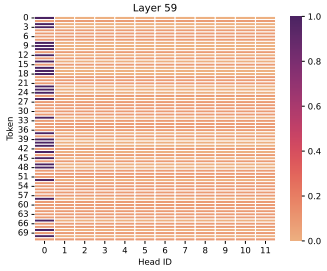
(lvii) Layer 56



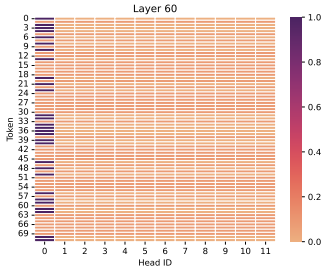
(lviii) Layer 57



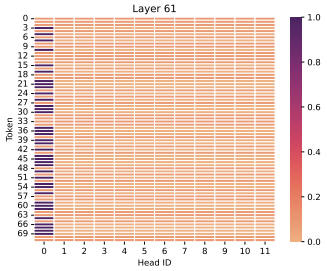
(lix) Layer 58



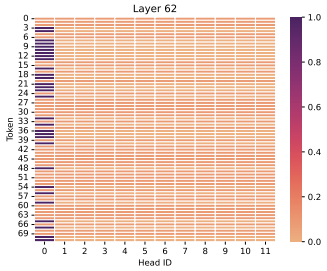
(lx) Layer 59



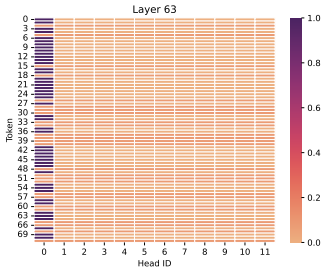
(lxi) Layer 60



(lxii) Layer 61

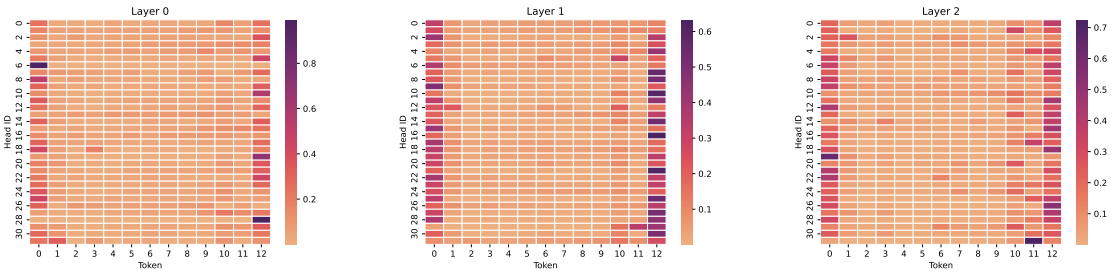


(lxiii) Layer 62

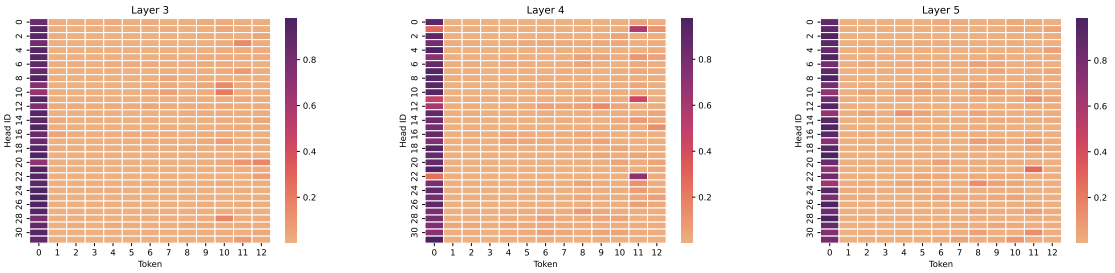


(lxiv) Layer 63

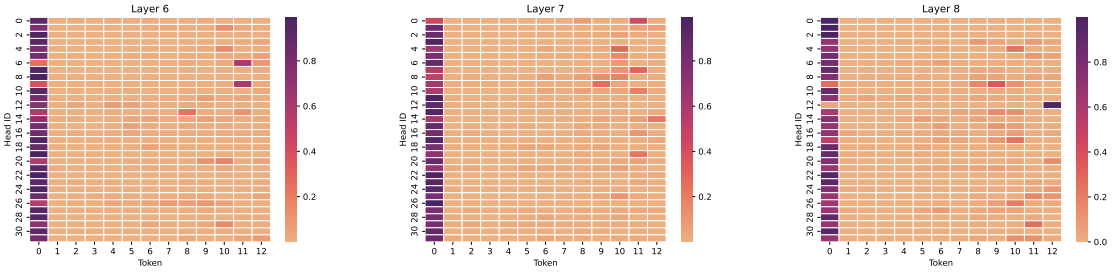
Figure 18. Layer Map OPT-66B



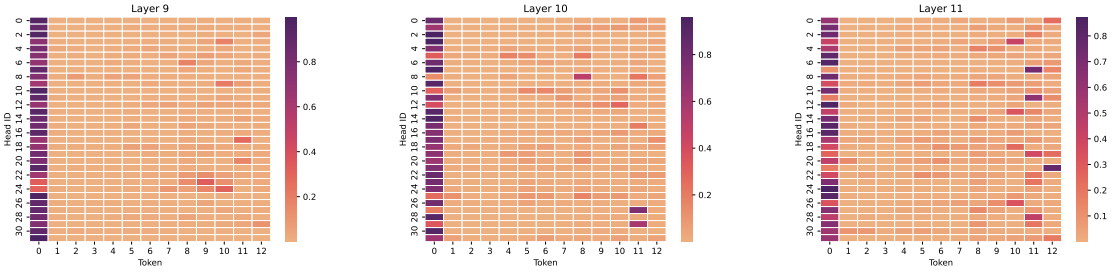
(a) Layer 0 (b) Layer 1 (c) Layer 2



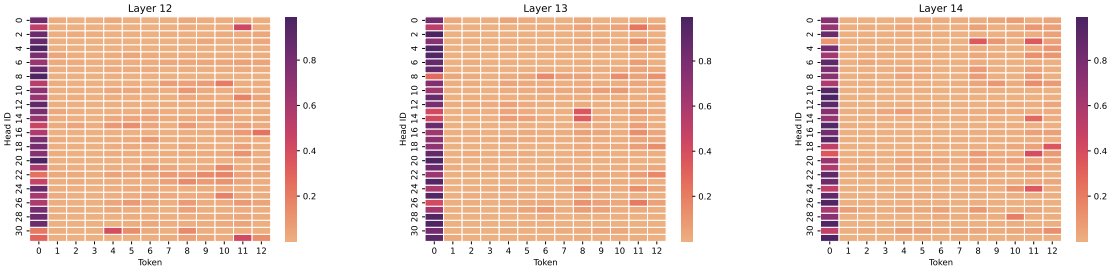
(d) Layer 3 (e) Layer 4 (f) Layer 5



(g) Layer 6 (h) Layer 7 (i) Layer 8

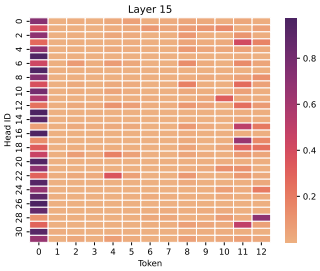


(j) Layer 9 (k) Layer 10 (l) Layer 11

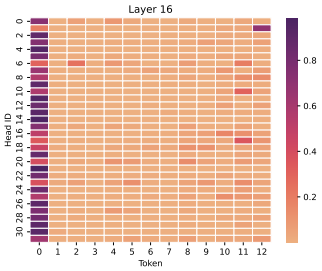


(m) Layer 12 (n) Layer 13 (o) Layer 14

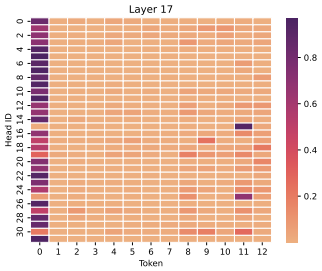
Figure 19. Activations of LLAMA-7B



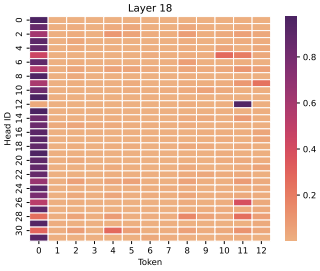
(xvi) Layer 15



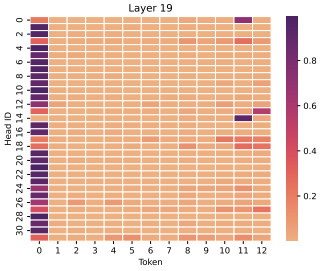
(xvii) Layer 16



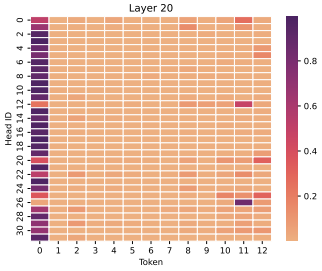
(xviii) Layer 17



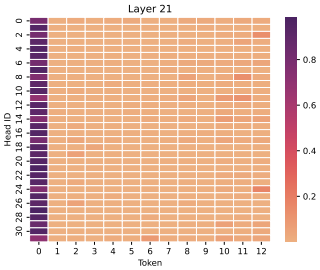
(xix) Layer 18



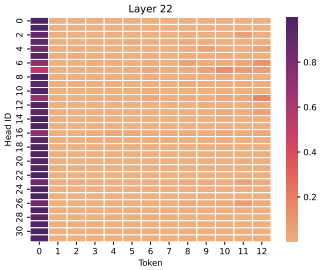
(xx) Layer 19



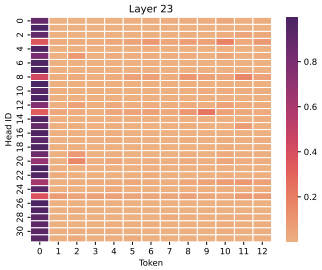
(xxi) Layer 20



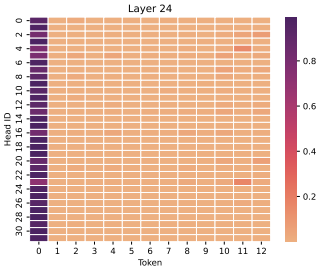
(xxii) Layer 21



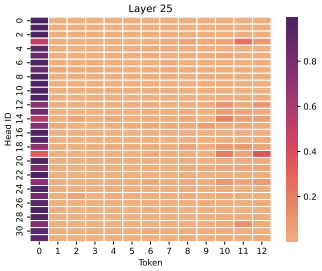
(xxiii) Layer 22



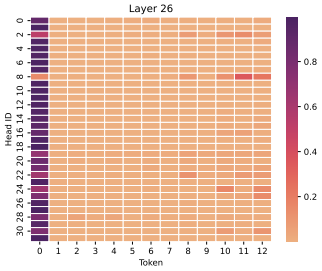
(xxiv) Layer 20



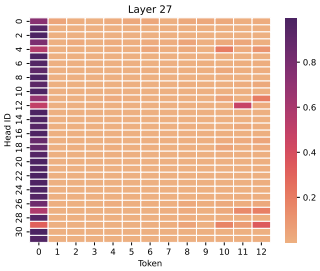
(xxv) Layer 24



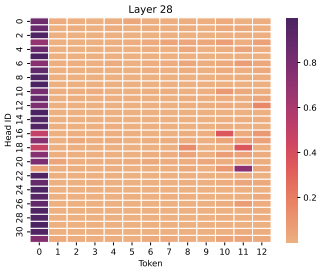
(xxvi) Layer 25



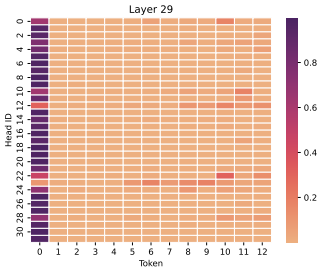
(xxvii) Layer 26



(xxviii) Layer 27

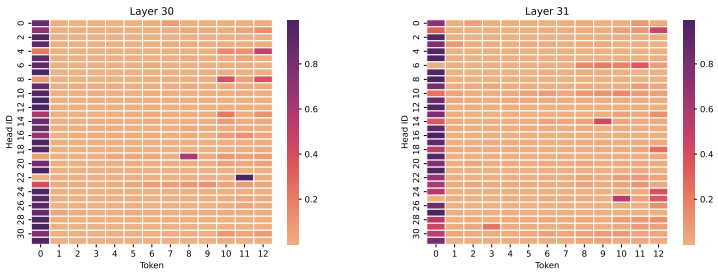


(xxix) Layer 28



(xxx) Layer 29

Figure 19. Activations of LLAMA-7B



(xxxi) Layer 30 (xxxii) Layer 31

Figure 19. Activations of LLAMA-7B