

PRUNING AGGREGATION PARAMETERS FOR LARGE LANGUAGE MODELS

Anonymous authors

Paper under double-blind review

ABSTRACT

Pruning is a highly effective approach for compressing large language models (LLMs). By strategically reducing model size, pruning significantly decreases both latency and GPU memory usage during inference, resulting in more efficient and cost-effective deployment of these models. Despite their effectiveness, current structured pruning algorithms have limitations. They still require extensive continued pre-training on large datasets to achieve model compression. Moreover, most of these methods are unable to reduce the memory usage of the key-value cache during generation tasks. In this work, we propose a novel pruning algorithm that requires no additional training and targets specific parameters within LLMs. We classify the model’s parameters into three categories: aggregation, transformation, and normalization. Our method primarily focuses on pruning the aggregation parameters in the higher layers of the model. To further improve the performance of the pruned LLM, we also introduce a rescaling parameter that adjusts the output of the pruned block. We conduct comprehensive experiments on a wide range of LLMs, including LLaMA3.1-8B/70B, Qwen2-7B/72B, Gemma2-9B, and Mistral-7B-v0.3. Our evaluation includes both generation and discriminative tasks across various benchmarks. The results consistently demonstrate that our method outperforms recent block pruning methods. This improvement is particularly notable in generation tasks, where our approach significantly outperforms existing baselines.

1 INTRODUCTION

Large language models (LLMs) (Touvron et al., 2023; OpenAI et al., 2023; Jiang et al., 2023; Yang et al., 2024; Gemma2-Team et al., 2024), pre-trained on extensive text data from across the internet, have achieved remarkable performance in downstream tasks such as information retrieval (Asai et al., 2024), code generation (Guo et al., 2024a), and mathematical reasoning (Wang et al., 2023; Yang et al., 2023; Huang et al., 2024). These LLMs, however, contain a huge number of parameters, resulting in substantially slower inference speed compared to their smaller counterparts. To address this issue in generation tasks, a common approach is to use key-value (KV) cache (Pope et al., 2023), which stores intermediate computation results. While this technique effectively trades space for time, speeding up inference, it also significantly increases GPU memory consumption. As reported in Zhou et al. (2024), the KV cache size can exceed the LLM model size during peak usage, and the inference latency increases as the KV cache size grows. As a result, one major bottleneck for LLM serving is GPU memory consumption.

Recent strategies to improve LLM efficiency primarily fall into two categories. The first category focuses on the models themselves, aiming to reduce inference latency and GPU memory consumption through pruning (Frantar & Alistarh, 2023; Ma et al., 2023; Jaiswal et al., 2023; Xia et al., 2024; Ashkboos et al., 2024; Xu et al., 2024; Jaiswal et al., 2024a; Zhang et al., 2024c; Dong et al., 2024c; Yin et al., 2024a;b; Zhao et al., 2024) or quantizing (Frantar et al., 2023; Xiao et al., 2023; Chee et al., 2023; Lin et al., 2024). The second category targets the KV cache, specifically for generation tasks, by either compressing (Dong et al., 2024b) or quantizing (Zhang et al., 2024d; Liu et al., 2024b) it to decrease GPU memory usage during inference. Among these approaches, structured pruning (Xia et al., 2024) searches for crucial substructures within the model while pruning other substructures through continued pretraining on extensive text datasets. However, a significant limitation of most current pruning algorithms is their inability to reduce the GPU memory consumption within the KV cache. To address this issue, KV cache compression algorithms like LESS (Dong et al., 2024b) have

054 been proposed, which maintain a constant-size KV cache by generating condensed representations of
 055 less important tokens. These approaches (Xia et al., 2024; Dong et al., 2024c), however, typically
 056 require designing specific learning objectives and loss functions, followed by extensive retraining
 057 of the base model on large text corpora to achieve the desired goal. We argue that these methods
 058 require an additional training phase, introducing significant computational overhead. Moreover, these
 059 approaches may struggle to maintain performance in domains not well-covered in the extra training
 060 data (Xia et al., 2024). This raises an important question: *Can we develop a training-free algorithm*
 061 *that effectively reduces GPU memory consumption with respect to the KV cache?* Our work addresses
 062 this challenge by drawing inspiration from an unexpected source: the intriguing connections between
 063 Graph Neural Networks (GNNs) (Kipf & Welling, 2016; 2017; Hamilton et al., 2017; Veličković
 064 et al., 2019) and LLMs. By exploring the parallels in their computation processes, we uncover
 065 insights that lead to a novel, training-free method for improving LLM efficiency.

066 Recent studies (Joshi, 2020; Ying et al., 2021;
 067 Kim et al., 2022; Nguyen et al., 2023; Barbero
 068 et al., 2024) have uncovered connections between
 069 GNNs and Transformers (Vaswani et al.,
 070 2017). The fundamental principle of GNNs is to
 071 aggregate information from neighboring nodes,
 072 resulting in smooth representations across the
 073 graph. This principle finds a parallel in LLMs,
 074 where the flow of contextual information can be
 075 conceptualized as a GNN operating on a fully
 076 connected graph, with connections governed by
 077 a causal attention mask. In this conceptualization,
 078 the process involves aggregating information
 079 from previous tokens to update the representations
 080 of subsequent ones. However, this aggregation
 081 process is not without challenges. In GNNs, while
 082 increasing the number of layers allows for the
 083 incorporation of higher-order neighbor information
 084 and potentially smoother representations, it also
 085 risks over-smoothing (Li et al., 2018). This
 086 phenomenon can lead to node representations
 087 converging to similar values, ultimately making
 088 them indistinguishable from one another. To
 089 address this issue in GNNs, GCNII (Chen et al.,
 090 2020a) has been developed, utilizing initial
 091 residual connections (Huang et al., 2017) and
 092 identity mappings, formulated as:

$$093 \mathbf{H}^{(\ell+1)} = \sigma \left(\left((1 - \alpha_\ell) \tilde{\mathbf{A}}\mathbf{H}^{(\ell)} + \alpha_\ell \mathbf{H}^{(0)} \right) \left((1 - \beta_\ell) \mathbf{I} + \beta_\ell \mathbf{H}^{(\ell)} \right) \right), \quad (1)$$

094 where $\tilde{\mathbf{A}} = \tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-1/2}$, \mathbf{A} is the adjacency matrix, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, $\tilde{\mathbf{D}}$ is the degree matrix of $\tilde{\mathbf{A}}$,
 095 and α_ℓ , β_ℓ , and $\mathbf{W}^{(\ell)}$ are the ℓ -th layer parameters. Although GCNII addresses over-smoothing, its
 096 accuracy improves by only 1.6% when increasing layers from 2 to 16 (Figure 1), at the cost of an
 097 eightfold increase in computation. Aggregation in GNNs is particularly computationally expensive,
 098 especially in large graphs, accounting for up to 90% of total training and inference time (Liu et al.,
 099 2023). GCNII can be made more efficient by reducing the number of aggregation operations during
 100 inference while keeping the training process unchanged. This modified version can be formulated as
 101 follows:

$$102 \mathbf{H}^{(\ell+1)} = \begin{cases} \sigma \left(\left((1 - \alpha_\ell) \tilde{\mathbf{A}}\mathbf{H}^{(\ell)} + \alpha_\ell \mathbf{H}^{(0)} \right) \left((1 - \beta_\ell) \mathbf{I} + \beta_\ell \mathbf{W}^{(\ell)} \right) \right) & \text{if } \ell \leq \frac{L}{2}, \\ \sigma \left(\left((1 - \alpha_\ell) \mathbf{H}^{(\ell)} + \alpha_\ell \mathbf{H}^{(0)} \right) \left((1 - \beta_\ell) \mathbf{I} + \beta_\ell \mathbf{W}^{(\ell)} \right) \right) & \text{if } \ell > \frac{L}{2}, \end{cases} \quad (2)$$

103 where L is the depth of GCNII. As shown in Figure 1, we achieve comparable performance to GCNII
 104 while halving the computational cost of aggregation during inference.

105 This phenomenon motivates us to explore whether a similar approach could be applied to LLMs.
 106 Similar to the computationally expensive aggregation in GNNs, the self-attention module in LLMs
 107 poses significant computational challenges. It exhibits quadratic time and memory complexity with

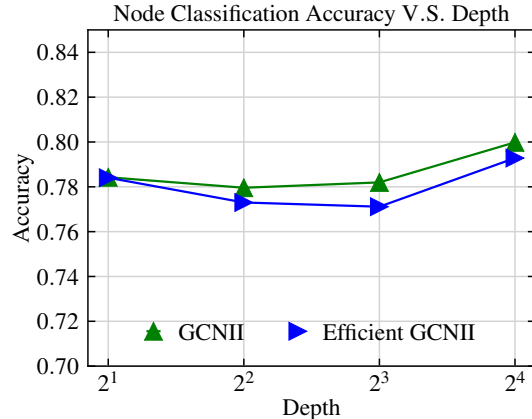


Figure 1: Performance comparison between GCNII and its efficient variant on the Pubmed dataset. The experiment evaluates both models with varying depths ($L = 2, 4, 8, \text{ and } 16$). The efficient GCNII demonstrates performance comparable to the original GCNII across all tested depths, despite its reduced computational complexity.

Algorithm 1 LLM inference with KV cache at each layer

-
- 1: **Input:** Current token’s hidden state $\mathbf{H}_t \in \mathbb{R}^{1 \times d}$, previous keys $\mathbf{K}_{\text{cache}} \in \mathbb{R}^{(t-1) \times (h \times d_k)}$, previous values $\mathbf{V}_{\text{cache}} \in \mathbb{R}^{(t-1) \times (h \times d_k)}$
 - 2: **Step 1. Self-Attention:**
 - 3: Apply Layer Normalization: $\mathbf{H}'_t = \mathbf{w}_1 \odot \left(\frac{\mathbf{H}_t}{\sqrt{\frac{1}{d} \sum_{i=1}^d ([\mathbf{H}_t]_{:,i})^2 + \epsilon}} \right)$, where $\mathbf{w}_1 \in \mathbb{R}^d$
 - 4: Apply Linear Projection: $\mathbf{Q}_t = \mathbf{H}'_t \mathbf{W}_Q$, $\mathbf{K}_t = \mathbf{H}'_t \mathbf{W}_K$, $\mathbf{V}_t = \mathbf{H}'_t \mathbf{W}_V$, where $\mathbf{W}_Q \in \mathbb{R}^{d \times (h \times g \times d_k)}$, $d = h \times g \times d_k$, and $\mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times (h \times d_k)}$
 - 5: Update KV Cache: $\mathbf{K}_{\text{cache}} \in \mathbb{R}^{t \times (h \times d_k)} \leftarrow \text{Concat}(\mathbf{K}_{\text{cache}}, \mathbf{K}_t)$, $\mathbf{V}_{\text{cache}} \in \mathbb{R}^{t \times (h \times d_k)} \leftarrow \text{Concat}(\mathbf{V}_{\text{cache}}, \mathbf{V}_t)$
 - 6: Reshape \mathbf{Q}_t , $\mathbf{K}_{\text{cache}}$, and $\mathbf{V}_{\text{cache}}$: $\mathbf{Q} \in \mathbb{R}^{1 \times h \times g \times d_k} \leftarrow \text{Reshape}(\mathbf{Q}_t)$, $\mathbf{K} \in \mathbb{R}^{t \times h \times d_k} \leftarrow \text{Reshape}(\mathbf{K}_{\text{cache}})$, $\mathbf{V} \in \mathbb{R}^{t \times h \times d_k} \leftarrow \text{Reshape}(\mathbf{V}_{\text{cache}})$
 - 7: Apply Rotary Position Embedding (RoPE): $\mathbf{Q} \leftarrow \text{RoPE}(\mathbf{Q})$, $\mathbf{K} \leftarrow \text{RoPE}(\mathbf{K})$, where RoPE is parameter-free
 - 8: Repeat \mathbf{K} and \mathbf{V} : $\mathbf{K}' \in \mathbb{R}^{t \times h \times g \times d_k} \leftarrow \text{Repeat}(\mathbf{K})$, $\mathbf{V}' \in \mathbb{R}^{t \times h \times g \times d_k} \leftarrow \text{Repeat}(\mathbf{V})$, where $\mathbf{K}'_{:,h_k:,g,:} = \mathbf{K}_{:,h_k,:}$ and $\mathbf{V}'_{:,h_k:,g,:} = \mathbf{V}_{:,h_k,:}$
 - 9: Compute Attention Scores $\mathbf{A} \in \mathbb{R}^{1 \times t \times h \times g}$: $\mathbf{A} = \text{softmax} \left(\frac{\mathbf{Q} \mathbf{K}'^T}{\sqrt{d_k}} \right)$, where $\mathbf{K}'^T \in \mathbb{R}^{t \times h \times d_k \times g}$
 - 10: Aggregate: $\mathbf{H}_{\text{attn}} = \mathbf{A} \mathbf{V}'$ through the second dimension ▷ Aggregation over the context
 - 11: Reshape \mathbf{H}_{attn} : $\mathbf{H}_{\text{attn}} \in \mathbb{R}^{1 \times d} \leftarrow \text{Reshape}(\mathbf{H}_{\text{attn}})$
 - 12: Add Residual Connection: $\mathbf{H}_t = \mathbf{H}_t + \mathbf{H}_{\text{attn}} \mathbf{W}_O$, where $\mathbf{W}_O \in \mathbb{R}^{d \times d}$
 - 13: **Step 2. Feedforward Network (FFN):**
 - 14: Apply Layer Normalization: $\mathbf{H}'_t = \mathbf{w}_2 \odot \left(\frac{\mathbf{H}_t}{\sqrt{\frac{1}{d} \sum_{i=1}^d ([\mathbf{H}_t]_{:,i})^2 + \epsilon}} \right)$, where $\mathbf{w}_2 \in \mathbb{R}^d$
 - 15: Apply FFN: $\mathbf{H}_{\text{FFN}} = (\sigma(\mathbf{H}'_t \mathbf{W}_{\text{gate}}) \odot (\mathbf{H}'_t \mathbf{W}_{\text{up}})) \mathbf{W}_{\text{down}}$, where σ is non-linear activation function, $\mathbf{W}_{\text{gate}}, \mathbf{W}_{\text{up}} \in \mathbb{R}^{d \times d_i}$, and $\mathbf{W}_{\text{down}} \in \mathbb{R}^{d_i \times d}$
 - 16: Add Residual Connection: $\mathbf{H}_t = \mathbf{H}_t + \mathbf{H}_{\text{FFN}}$
 - 17: **Output:** Updated keys $\mathbf{K}_{\text{cache}}$, updated values $\mathbf{V}_{\text{cache}}$, and updated \mathbf{H}_t
-

respect to sequence length (Dao et al., 2022), making it a bottleneck in LLM serving. In this work, inspired by the efficient version of GCNII discussed earlier, we propose a training-free pruning strategy called AggregationPruner that targets only the query and key parameters in the higher layers of LLMs. By selectively pruning these aggregation parameters, AggregationPruner compresses the model and achieves a significant reduction in GPU memory consumption associated with the KV cache during generation tasks. Considering the complex and black-box nature of LLMs, our approach carefully avoids pruning transformation or normalization parameters to minimize the potential negative impacts of pruning on downstream tasks. To further enhance performance, we incorporate a rescaling parameter for the output of pruned blocks. Extensive experiments demonstrate that our method outperforms recent block pruning algorithms (Men et al., 2024; Zhong et al., 2024; Gromov et al., 2024; He et al., 2024; Siddiqui et al., 2024; Liu et al., 2024a; Zhang et al., 2024a; Jaiswal et al., 2024b; Chen et al., 2024; Kim et al., 2024) across a wide range of downstream tasks and testing LLMs. Notably, our approach shows significant performance improvement in generation tasks while maintaining the same memory consumption with Self-AttentionPruner and LayerPruner (Gromov et al., 2024; He et al., 2024) during inference.

2 PRELIMINARIES

2.1 DECODER-ONLY LARGE LANGUAGE MODEL

In decoder-only LLMs, information flows through self-attention modules, with each token aggregating context from all preceding tokens in the sequence. This autoregressive process enables the model to generate each subsequent token based on the information from earlier tokens. To preserve the causal structure of language generation during training, attention is masked, preventing tokens from accessing information from future positions in the sequence. For a given input sentence

Table 1: Classification of LLM layer parameters based on their functional roles.

Type	w_1	W_Q	W_K	W_V	W_O	w_2	W_{gate}	W_{up}	W_{down}
Aggregation Parameter		✓	✓						
Transformation Parameter				✓	✓		✓	✓	✓
Normalization Parameter	✓					✓			

$\mathbf{x} = \{x_1, \dots, x_n\}$, LLMs employ a standard language modeling objective. This objective aims to maximize the following: (Radford et al., 2018):

$$L(\mathbf{x}) = \sum_i \log P(x_i | x_{i-1}, \dots, x_1; \Theta), \quad (3)$$

where $P(x_i | x_{i-1}, \dots, x_1; \Theta)$ represents the probability of token x_i given all preceding tokens and the model parameters Θ . Conceptually, LLMs can be viewed as operating on a complete graph structure, with tokens serving as nodes and attention scores as edges. Both LLMs and GNNs employ a similar strategy for information processing: they iteratively refine representations by incorporating contextual information. In LLMs, this context is derived from preceding tokens in a sequence, while in GNNs, it comes from neighboring nodes in a graph. Despite operating in different domains, these two model types share a fundamental approach to information aggregation and propagation. This shared mechanism allows both LLMs and GNNs to generate context-aware representations.

Algorithm 1 illustrates the inference computation process of a decoder-only LLM layer. We categorize the model parameters into three functional groups: aggregation, transformation, and normalization, as detailed in Table 1. Aggregation parameters, such as W_Q and W_K , are used to compute attention scores within the adjacency matrix \mathbf{A} . These parameters enable the model to aggregate information from preceding tokens, integrating context and capturing dependencies among tokens. Transformation parameters, such as W_V , W_O , and W_{gate} , apply linear transformations and feedforward operations to the hidden states of tokens. These parameters are crucial for the model’s ability to process input and generate output. Normalization parameters, like w_1 and w_2 , play a significant role in stabilizing the training process. By maintaining a consistent scale in the output, they help prevent issues such as vanishing or exploding gradients. In this work, we propose a pruning algorithm that specifically targets the aggregation parameters to improve the LLMs’ efficiency.

2.2 BLOCK PRUNING STRATEGIES

Recent research (Men et al., 2024; Zhong et al., 2024; Gromov et al., 2024; He et al., 2024; Siddiqui et al., 2024) has revealed the presence of redundant parameters in the higher layers of LLMs. These studies demonstrate that selectively pruning certain blocks within these higher layers has little performance degradation on downstream discriminative tasks. These pruning strategies can be classified into three distinct approaches: Self-AttentionPruner, FFNPruner, and LayerPruner. Each targets different components of the model:

- **Self-AttentionPruner:** This method bypasses the self-attention computation module (Step 1 in Algorithm 1), removing parameters across all three categories: Aggregation, Transformation, and Normalization.
- **FFNPruner:** By skipping the feed-forward network computation process (Step 2), this approach primarily prunes Transformation and Normalization parameters.
- **LayerPruner:** This method skips an entire layer, resulting in the removal of all parameter types within that layer.

These studies have introduced heuristic metrics to evaluate the importance of blocks within each layer of LLMs. A consistent finding across these works is the greater significance of parameters in lower layers compared to those in higher layers. Consequently, pruning algorithms typically target parameters in higher layers while preserving those in lower layers.

This phenomenon can be intuitively explained through the lens of GNNs. The fundamental principle of GNNs is to aggregate information from neighboring nodes to achieve smoother representations. However, as the number of GNN layers increases, node representations tend to converge towards a

Algorithm 2 LLM inference with AggregationPruner at each layer

-
- 1: **Input:** Current token’s hidden state $\mathbf{H}_t \in \mathbb{R}^{1 \times d}$
 - 2: **Step 1. Transformation without Self-Attention:**
 - 3: Apply Layer Normalization as Line 3 in Algorithm 1
 - 4: Apply Linear Projection: $\mathbf{V}_t = \mathbf{H}_t \mathbf{W}_V$, where $\mathbf{W}_V \in \mathbb{R}^{d \times (h \times d_k)}$
 - 5: Reshape \mathbf{V}_t : $\mathbf{V} \in \mathbb{R}^{1 \times h \times d_k} \leftarrow \text{Reshape}(\mathbf{V}_t)$
 - 6: Repeat \mathbf{V} : $\mathbf{V}' \in \mathbb{R}^{1 \times h \times g \times d_k} \leftarrow \text{Repeat}(\mathbf{V})$, where $\mathbf{V}'_{0, h_k, :, g, :} = \mathbf{V}_{0, h_k, :, :}$
 - 7: Reshape \mathbf{V}' : $\mathbf{V}_t \in \mathbb{R}^{1 \times d} \leftarrow \text{Reshape}(\mathbf{V}')$
 - 8: Add Residual Connection: $\mathbf{H}_t = \mathbf{H}_t + \boxed{\alpha \mathbf{V}_t \mathbf{W}_O}$ ▷ Introduce a rescaling parameter
 - 9: **Step 2. Feedforward Network (FFN):**
 - 10: Apply FFN and Residual Connection as Lines 14-16 in Algorithm 1
 - 11: **Output:** Updated \mathbf{H}_t
-

common value. Beyond a certain point, adding more layers contributes minimally to changing node representations, which can be formulated as follows:

$$\lim_{\ell \rightarrow \infty} \left\| \mathbf{H}^{(\ell+1)} - \mathbf{H}^{(\ell)} \right\|_F^2 = \lim_{\ell \rightarrow \infty} \left\| \mathbf{P}^{(\ell+1)} \mathbf{X} - \mathbf{P}^{(\ell)} \mathbf{X} \right\|_F^2 = 0. \quad (4)$$

While the propagation matrix \mathbf{P} in GNNs is static, the attention matrix in LLMs is dynamic. Despite this difference, recent studies (Shi et al., 2022; Nguyen et al., 2023) have revealed that Transformers can also experience over-smoothing, similar to GNNs. This phenomenon provides insight into the behavior of the Block Importance (BI) metric proposed by Men et al. (2024):

$$\text{BI}^{(\ell)} = 1 - \mathbb{E}_{\mathbf{H}_t} \frac{\mathbf{H}_t^{(\ell)} \cdot \mathbf{H}_t^{(\ell+1)}}{\left\| \mathbf{H}_t^{(\ell)} \right\|_2 \left\| \mathbf{H}_t^{(\ell+1)} \right\|_2}. \quad (5)$$

The BI metric tends to decrease as the layer index ℓ increases. This observation explains why recent pruning algorithms target blocks in higher layers: these layers contribute less unique information. Informed by these insights, our work also focuses on pruning aggregation parameters in the higher layers of LLMs.

3 AGGREGATIONPRUNER

In this section, we first discuss the motivation behind our proposed AggregationPruner in Section 3.1 and 3.2. Then, we provide the details of our pruning algorithm in Section 3.3.

3.1 THE BOTTLENECK IN LLM SERVING

In applications such as chatbots and content generation tools, which handle a high volume of daily API requests, maintaining low latency is crucial. This is typically achieved by batching multiple requests for inference, thereby reducing computational waste. Moreover, modern LLMs employ the KV cache to accelerate inference by storing intermediate results. While effective, this approach leads to increased memory consumption as the number of requests grows. To illustrate the scale of memory consumption from the KV cache, we use an example from PagedAttention (Kwon et al., 2023). A 13B parameter OPT model (Zhang et al., 2022), capable of generating up to 2048 tokens, requires approximately 800 KB of GPU memory per token. This can lead to a potential consumption of 1.6 GB per request. Given that LLM operations are primarily constrained by memory bandwidth (Dao et al., 2022), the amount of memory access becomes the primary factor in determining runtime. Consequently, understanding the mechanism by which LLMs generate and utilize the KV cache is essential for optimizing resource utilization.

3.2 DISTINCT ROLES OF PARAMETER TYPES IN LARGE LANGUAGE MODELS

3.2.1 THE ROLE OF AGGREGATION PARAMETER

As previously discussed, aggregation parameters play a crucial role in calculating attention scores, which are essential for aggregating contextual information from preceding tokens to subsequent ones.

Algorithm 3 Top-down α grid search for AggregationPruner

```

270 1: Input: Pruned LLM, number of pruned layers  $P$ , alpha search range  $0, 0.1, 0.2, \dots, 1.0$ 
271 2: Initialize:  $\alpha_{best} = 1.0^P$  (list of length  $P$ )
272 3: for  $\ell = L$  to  $L - P + 1$  step  $-1$  do ▷ Start from the top layer
273 4:    $PPL_{best} = \infty$ 
274 5:   for  $\alpha \in 0, 0.1, 0.2, \dots, 1.0$  do
275 6:     Set  $\alpha_\ell = \alpha$  in the pruned LLM
276 7:      $PPL = \text{Perplexity}(\text{LLM}_{\text{pruned}}(\alpha_{best}))$ 
277 8:     if  $PPL < PPL_{best}$  then
278 9:        $PPL_{best} = PPL$ 
279 10:       $\alpha_{best}[\ell] = \alpha$ 
280 11:     end if
281 12:   end for
282 13: end for
283 14: Output: Optimal  $\alpha_{best}$  for each pruned layer

```

This process involves computing the inner product of queries and keys, resulting in quadratic time and memory complexity with respect to sequence length. To accelerate the generation of subsequent tokens, modern LLMs typically employ a KV cache mechanism as illustrated in Algorithm 1. This approach stores previously calculated keys and values, thereby reducing computational overhead. When generating a new token, the model only needs to compute the query, key, and value for the last token in the sequence. It then combines the KV cache with the last token’s query and key to aggregate information from previous tokens, integrating this context into the last token’s representation. By avoiding the need to recompute keys for each token, this approach significantly accelerates the calculation of attention scores. These scores are then used to aggregate contextual information from the V cache and the last token’s value. This optimization strategy greatly enhances inference speed by minimizing redundant computations, particularly for long sequence generation tasks.

While the KV cache significantly accelerates inference, it also introduces substantial GPU memory consumption. As previously discussed, higher layers in LLMs typically contribute less unique information to the model’s output. Leveraging this insight, our work focuses on pruning aggregation parameters in these higher layers to reduce the size of the KV cache. This approach aims to balance the trade-off between inference speed and memory efficiency, optimizing overall model performance.

3.2.2 THE ROLE OF TRANSFORMATION PARAMETERS

Transformation parameters in LLMs comprise two main components: \mathbf{W}_v and \mathbf{W}_o in the Self-Attention module, and \mathbf{W}_{gate} , \mathbf{W}_{down} , \mathbf{W}_{up} in the Feed-Forward Network (FFN). These parameters apply linear transformations on token embeddings and, as some research (Anderson, 1972; Kohonen, 1972; Geva et al., 2021; Meng et al., 2023) suggests, serve as storage of compressed knowledge (Deleang et al., 2024; Lester et al., 2024) derived from vast internet-scale text data.

Current block pruner methods risk discarding valuable stored knowledge when pruning these transformation parameters. Furthermore, since pruning aggregation parameters already provides substantial memory savings, further pruning of transformation parameters results in diminishing returns. This additional pruning could also introduce potential issues, especially when it comes to maintaining performance across various downstream tasks.

3.3 OUR PROPOSED PRUNING ALGORITHM: AGGREGATIONPRUNER

Building on the insights discussed above, we introduce AggregationPruner, a novel pruning algorithm designed specifically for LLMs. This approach strategically focuses on pruning only the aggregation parameters in the higher layers of LLMs, preserving the knowledge-rich transformation parameters. By doing so, AggregationPruner achieves substantial memory efficiency gains while preserving the model’s core knowledge base. The computation process for the higher layers, incorporating our pruning strategy, is detailed in Algorithm 2. It’s important to note that many LLMs employ Grouped-query attention (GQA) (Ainslie et al., 2023). Therefore, a modification is required to accommodate this architecture as shown in Algorithm 2. Specifically, in Line 6, we must replicate the

Algorithm 4 LLM inference with AggregationPruner

```

1: Input: Token sequence  $\mathcal{T}$ , the number of layers  $L$ , the number of pruning layers  $P$ 
2: for  $\ell = 1$  to  $L$  do
3:   if  $\ell \leq L - P$  then
4:     Run Algorithm 1 on  $\mathcal{T}$ : Compute  $\mathbf{H}^{(\ell)}$ 
5:   else
6:     Run Algorithm 2 on  $\mathcal{T}$ : Compute  $\mathbf{H}^{(\ell)}$ 
7:   end if
8: end for
9: Output: Final token representation  $\mathbf{H}^{(L)}$ 

```

value matrix \mathbf{V} a total of $g - 1$ times, where g represents the number of query groups. This replication ensures compatibility with the GQA mechanism, enabling our AggregationPruner to seamlessly integrate with modern LLM architectures.

When modifying the higher layers of an LLM, we propose that the original residual connection coefficient of 1 may no longer be optimal. Inspired by GCNII, which uses a decreasing coefficient to address the diminishing unique information in higher layers caused by over-smoothing, we introduce a rescaling parameter. This parameter, denoted as α , adjusts the pruned block’s output within the residual connection, as shown in Line 8 of Algorithm 2.

Determining the optimal value for α presents a challenge. Traditional retraining methods are not applicable due to α ’s non-differentiable nature. While some recent works have employed Zeroth-Order Optimization (Guo et al., 2024b; Zhang et al., 2024b) to estimate gradients during fine-tuning, we propose a simpler, more efficient approach: a greedy search strategy. Our method involves calculating the perplexity of the pruned LLM to identify the optimal α . To simplify the search process, we adopt a top-down approach. We begin by determining α for the uppermost layer and then use this value as a starting point for the subsequent layer. This cascading strategy significantly reduces the search space. The entire process is implemented as a grid search as illustrated in Algorithm 3, balancing efficiency with thoroughness. This approach allows us to fine-tune the rescaling parameter across layers, optimizing the model’s performance post-pruning without the need for extensive retraining. The development of more complex search strategies leaves room for future work. The inference process, which incorporates AggregationPruner, is detailed in Algorithm 4.

4 EXPERIMENTS

In this section, we present a comprehensive evaluation of our proposed pruning algorithm, assessing its performance across six LLMs and ten diverse benchmarks. By conducting experiments on various LLMs and benchmarks, we aim to establish consistent and reliable results.

4.1 SETUP

Baselines. We evaluate AggregationPruner against three baselines: FNNPruner, LayerPruner, and Self-AttentionPruner, which are described in Section 2.2. These baseline methods employ various heuristic metrics to determine which layers should be pruned. While there may be minor variations in the specific layers selected for pruning, all these methods generally prune from top to bottom. In this work, we evaluate performance by pruning different blocks within the selected layers. To ensure a fair comparison, we apply a top-to-bottom pruning approach for all methods as shown in Algorithm 4.

Testing LLMs. We evaluate all pruning algorithms on 6 LLMs: LLaMA3.1-8B/70B (Touvron et al., 2023), Qwen2-7B/72B (Yang et al., 2024), Gemma2-9B (Gemma2-Team et al., 2024), and Mistral-7B-v0.3 (Jiang et al., 2023). All experiments are conducted using Nvidia H100/A100 80G GPUs. However, due to memory constraints, we are unable to load the full weights of LLaMA3.1-70B and Qwen2-72B directly onto the GPU. To address this limitation, we employ the bnb quantization method provided by Hugging Face to compress these two models to 4-bit precision for our experiments.

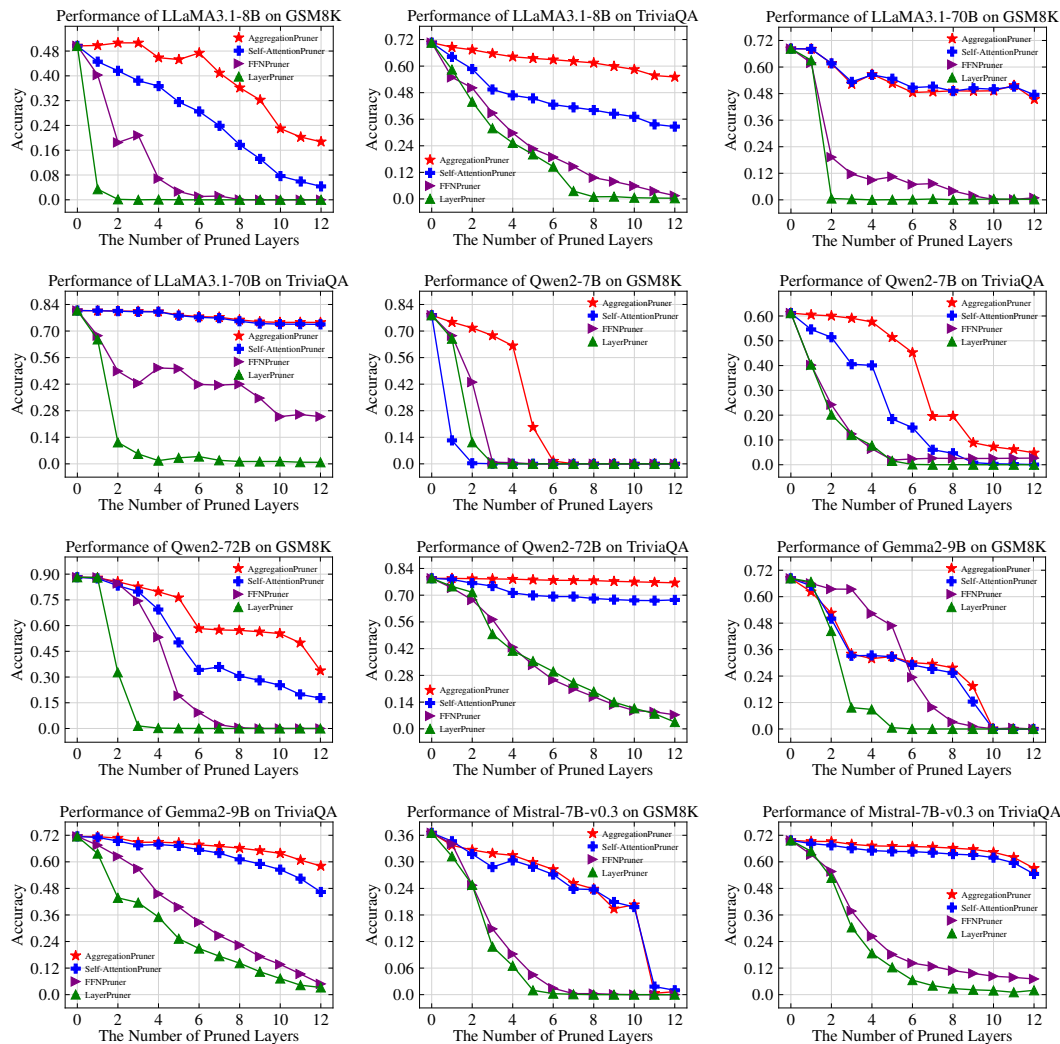


Figure 2: We evaluate the performance of 6 LLMs, including LLaMA3.1-8B/70B, Qwen2-7B/72B, Gemma2-9B, and Mistral-7B-v0.3, on generation tasks such as GSM8K and TriviaQA. Our evaluation involves progressively pruning layers, starting from 0 and extending up to 12 layers.

Benchmarks. We employ the `lm-evaluation-harness` package (Gao et al., 2021) to conduct experiments on both generation and discriminative tasks. While current LLM deployments primarily focus on generation tasks, discriminative tasks are often used to evaluate overall model performance. It’s important to note that LLMs only generate the KV cache during generation tasks. Discriminative tasks, on the other hand, involve providing inputs and directly obtaining results, such as classification labels or regression values. Our generation tasks include 5-shot GSM8K (Cobbe et al., 2021) and 5-shot TriviaQA (Joshi et al., 2017). For discriminative tasks, we use 7-shot CommonsenseQA (Talmor et al., 2019), 5-shot WinoGrande (Sakaguchi et al., 2019), 25-shot ARC-Challenge (Clark et al., 2018), 0-shot BoolQ (Clark et al., 2019), 0-shot OpenBookQA (Mihaylov et al., 2018), 0-shot PIQA (Bisk et al., 2020), 0-shot MedQA (Jin et al., 2020), and 5-shot MMLU (Hendrycks et al., 2021). We report the accuracy for these tasks as recommended by the `lm-evaluation-harness` package.

4.2 RESULTS

Generation Tasks. Figure 2 presents the results for all pruning algorithms, showing the superior performance of our proposed AggregationPruner across multiple generation tasks and language models. Our method outperforms the baselines on both generation tasks with LLaMA3.1-8B, Qwen2-

Table 2: The Performance of LLaMA3.1-8B on Discriminative Tasks.

LLaMA3.1-8B										
#Layers	Method	CommonSenseQA	WinoGrande	ARC-Challenge	BoolQ	OpenBookQA	PIQA	MedQA	MMLU	Average
0	No Pruning	73.6	77.2	54.7	82.1	33.4	80.0	59.9	65.2	65.8
2	FFNPruner	73.3	68.7	48.1	81.1	35.2	76.2	59.7	65.1	63.4
	LayerPruner	73.4	67.2	47.5	81.1	37.6	75.6	59.9	65.1	63.4
	Self-AttentionPruner	71.2	77.8	50.4	75.9	31.0	78.8	60.2	62.1	63.4
	AggregationPruner	73.9	78.0	53.4	81.7	32.6	79.7	60.0	64.9	65.5
4	FFNPruner	73.3	66.2	45.1	77.4	32.2	75.0	53.9	62.6	60.7
	LayerPruner	71.7	65.7	45.3	78.1	34.0	74.1	57.3	63.4	61.2
	Self-AttentionPruner	71.1	76.7	49.5	56.6	30.4	77.9	60.4	61.6	60.5
	AggregationPruner	74.4	77.5	52.7	78.0	33.0	79.1	60.1	65.0	65.0
6	FFNPruner	71.2	65.6	41.2	71.2	30.0	71.3	46.3	56.0	56.6
	LayerPruner	72.4	60.9	43.9	79.3	33.4	71.9	53.5	61.3	59.6
	Self-AttentionPruner	71.0	77.4	50.3	52.4	30.6	77.8	59.0	62.1	60.1
	AggregationPruner	74.3	77.7	53.2	75.7	32.6	78.9	59.2	64.8	64.6
8	FFNPruner	72.6	64.7	37.5	62.2	27.2	68.6	55.3	62.8	56.4
	LayerPruner	61.9	62.3	41.0	62.3	30.4	69.8	53.2	54.5	54.4
	Self-AttentionPruner	71.6	76.6	49.1	51.8	30.4	77.7	58.4	62.0	59.7
	AggregationPruner	74.0	77.7	53.6	74.8	33.2	79.0	59.9	64.8	64.6
10	FFNPruner	71.6	63.9	32.2	62.1	24.4	65.6	53.6	61.3	54.3
	LayerPruner	63.9	61.6	36.7	62.3	26.6	68.4	57.4	62.4	54.9
	Self-AttentionPruner	69.9	76.6	47.7	50.8	30.6	77.3	58.0	61.7	59.1
	AggregationPruner	74.4	78.3	52.4	74.8	32.2	78.8	60.7	64.5	64.5
12	FFNPruner	72.7	62.0	31.2	63.1	21.8	63.8	57.9	63.2	54.5
	LayerPruner	63.6	58.2	34.0	63.3	23.6	64.6	49.0	54.7	51.4
	Self-AttentionPruner	70.4	75.5	45.1	49.4	29.0	75.5	57.5	61.7	58.0
	AggregationPruner	74.3	77.1	51.3	75.0	31.0	77.3	60.8	64.5	63.9
14	FFNPruner	71.4	62.4	29.2	62.4	18.8	61.2	59.8	62.2	53.4
	LayerPruner	71.8	58.7	32.1	62.2	24.4	63.1	59.6	64.2	54.5
	Self-AttentionPruner	67.6	75.3	44.0	49.8	27.0	75.4	57.3	60.3	57.1
	AggregationPruner	72.4	76.7	49.1	76.3	29.6	77.5	61.7	64.6	63.5

7B/72B, and Gemma2-9B for TriviaQA. For Mistral-7B-v0.3 on TriviaQA and LLaMA3.1-70B, it shows a slight improvement. Additionally, on GSM8K with LLaMA3.1-70B, Gemma2-9B, and Mistral-7B-v0.3, our performance is comparable to the best baseline. These results consistently demonstrate that our method surpasses the three baselines across various models and tasks. Besides, our results reveal a clear ranking in overall performance among the pruning methods: AggregationPruner > Self-AttentionPruner > FFNPruner > LayerPruner. Notably, FFNPruner and LayerPruner exhibit a rapid decline in performance, dropping to zero as the number of pruned layers increases, compared with the other two methods. These results emphasize the critical importance of transformation parameters in both the FFN and Self-Attention modules for generation tasks. This observation aligns with our claim in Section 3.2.2.

Furthermore, our analysis revealed that as the number of pruned layers increases, the performance of LLMs drops more rapidly on GSM8K compared to TriviaQA. This discrepancy can be attributed to the differing response lengths required for each task. We observed that unpruned LLMs typically encounter the end-of-sequence (EOS) token within 16 tokens when generating answers for TriviaQA. In contrast, GSM8K often requires more (up to 256) tokens to produce a complete answer. Pruned LLMs, which generate one token at a time, are more susceptible to errors than their unpruned counterparts. This vulnerability is exacerbated in tasks requiring longer responses, as each additional token introduces the potential for error accumulation. Consequently, the extended response length needed for GSM8K leads to a more pronounced performance decline in pruned LLMs compared to the shorter responses typical of TriviaQA.

Discriminative Tasks. We present the performance of six LLMs on discriminative tasks in Tables 2, 5, 6, and 4. Due to space limit in the main text, Tables 5, 6, and 4 are included in Appendix C. We also report the average performance across eight discriminative tasks. The results demonstrate that our pruning algorithm outperforms the baselines on LLaMa3.1-8B, Qwen2-7B/72B, and Mistral-7B-v0.3, while achieving comparable performance to the best baseline on LLaMa3.1-70B and Gemma2-9B. Notably, as we increase the number of pruned layers, the performance degradation on discriminative tasks is less pronounced compared to generation tasks. This discrepancy can be

486 attributed to the nature of discriminative tasks, which typically involve multiple-choice questions with
 487 limited options, making them inherently simpler than generation tasks that require predicting the next
 488 token from the entire vocabulary. To further validate this claim, we conduct additional experiments
 489 using a reward model with AggregationPruner. Specifically, we evaluate the Skywork/Skywork-
 490 Reward-Llama-3.1-8B model from Hugging Face on RewardBench (Lambert et al., 2024) to assess
 491 the impact of pruning algorithm on reward model performance.

492 Figure 8 in Appendix C illustrates the performance of the reward model on RewardBench. Notably, when 16 layers
 493 are pruned, the model’s performance remains nearly identical to that of the unpruned version. However, an additional
 494 experiment reveals differences when using the pruned and unpruned models to annotate rewards for online alignment
 495 (Cen et al., 2024; Dong et al., 2024a). We observe a significant disparity in the reward distributions generated
 496 by the pruned and unpruned models. The mean in the rewards gap is 3.53, with a standard deviation of 5.77. This
 497 discrepancy can be attributed to the nature of the tasks: RewardBench primarily involves preference choices between
 498 two responses, essentially a binary classification problem. In contrast, reward annotation operates on a continuous
 499 real number scale, which is a more challenging task. These findings lead us to conclude that pruned models are better
 500 suited for maintaining performance on discriminative tasks with limited options. This conclusion makes pruned
 501 reward models particularly well-suited for online Direct Preference Optimization (DPO) (Rafailov et al., 2023)
 502 settings. In such settings, each iteration requires only on-policy preference data, and the reduced latency of pruned
 503 models is advantageous. However, this same attribute makes them less ideal for online Reinforcement Learning
 504 from Human Feedback (RLHF) (Ouyang et al., 2022), where more nuanced reward annotations may be necessary.

514 4.3 ABLATION STUDY

515
 516 In this section, we evaluate the efficacy of our proposed α search algorithm, as described in Section 3.3. Our
 517 experiments focus on Qwen2-7B, and we present the average accuracy across eight discriminative tasks. As
 518 illustrated in Figure 3, the alpha value obtained through our grid search method demonstrates better performance
 519 compared to the default setting of $\alpha = 1$. These results demonstrate the effectiveness of our algorithm in
 520 improving model performance.

522 5 RELATED WORK

523
 524 **Pruning.** Pruning is a widely adopted and efficient technique in both Computer Vision and Large
 525 Language Models. It can be categorized into two main types: Structured Pruning and Unstructured Pruning.
 526 Structured Pruning (Lagunas et al., 2021; Xia et al., 2022; Kurtic et al., 2023; He & Xiao, 2023; Xia et al., 2024)
 527 involves removing entire filters from neural networks, making it particularly conducive to model deployment.
 528 On the other hand, Unstructured Pruning (Chen et al., 2020b; Sanh et al., 2020) focuses on removing individual
 529 neurons within the network. Some recent works (Men et al., 2024; Zhong et al., 2024; Gromov et al., 2024; He et al., 2024; Siddiqui et al., 2024)
 530 have been proposed to prune blocks in the higher layers of Large Language Models.

533 6 CONCLUSION

534
 535 In this work, we propose AggregationPruner, a novel approach that focuses on pruning query and key parameters
 536 in the higher layers of LLMs. Our method can reduce GPU memory consumption associated with the KV cache
 537 during generation tasks. Through extensive experimentation, we demonstrate that our pruning algorithm
 538 consistently outperforms recent block pruning techniques, offering a significant advancement in model
 539 efficiency without compromising performance. We hope our work will inspire future research on pruning
 strategies to reduce the KV cache in LLM serving.

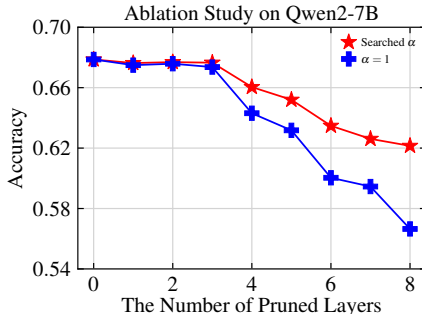


Figure 3: Performance comparison between the default alpha setting ($\alpha = 1$) and the α value obtained through grid search for Qwen2-7B. The experiment evaluates model accuracy averaged across eight discriminative tasks.

REFERENCES

- 540
541
542 Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit
543 Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints.
544 In *Empirical Methods in Natural Language Processing*, 2023.
- 545 James A Anderson. A simple neural network generating an interactive memory. *Mathematical*
546 *Biosciences*, 14(3-4):197–220, 1972.
- 547
548 Akari Asai, Zexuan Zhong, Danqi Chen, Pang Wei Koh, Luke Zettlemoyer, Hannaneh Hajishirzi, and
549 Wen-tau Yih. Reliable, adaptable, and attributable language models with retrieval. *arXiv preprint*
550 *2403.03187*, 2024.
- 551 Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and
552 James Hensman. SliceGPT: Compress large language models by deleting rows and columns. In
553 *International Conference on Learning Representations*, 2024.
- 554
555 Federico Barbero, Andrea Banino, Steven Kapturowski, Dharshan Kumaran, João GM Araújo,
556 Alex Vitvitskiy, Razvan Pascanu, and Petar Veličković. Transformers need glasses! information
557 over-squashing in language tasks. *arXiv preprint arXiv:2406.04267*, 2024.
- 558 Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical
559 commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*,
560 2020.
- 561
562 Shicong Cen, Jincheng Mei, Katayoon Goshvadi, Hanjun Dai, Tong Yang, Sherry Yang, Dale
563 Schuurmans, Yuejie Chi, and Bo Dai. Value-incentivized preference optimization: A unified
564 approach to online and offline rlhf. *arXiv preprint arXiv:2405.19320*, 2024.
- 565 Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. Quip: 2-bit quantization of
566 large language models with guarantees. In *Advances in Neural Information Processing Systems*,
567 2023.
- 568
569 Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph
570 convolutional networks. In *International Conference on Machine Learning*, 2020a.
- 571
572 Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and
573 Michael Carbin. The lottery ticket hypothesis for pre-trained bert networks. In *Advances in Neural*
574 *Information Processing Systems*, 2020b.
- 575
576 Xiaodong Chen, Yuxuan Hu, and Jing Zhang. Compressing large language models by streamlining
577 the unimportant layer. *arXiv preprint arXiv:2403.19135*, 2024.
- 578
579 Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina
580 Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *North*
581 *American Association for Computational Linguistics*, 2019.
- 582
583 Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and
584 Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge.
585 *arXiv preprint arXiv:1803.05457*, 2018.
- 586
587 Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser,
588 Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve
589 math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- 590
591 Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-
592 efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems*,
593 2022.
- 594
595 Gregoire Deletang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher
596 Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus
597 Hutter, and Joel Veness. Language modeling is compression. In *International Conference on*
598 *Learning Representations*, 2024.

- 594 Hanze Dong, Wei Xiong, Bo Pang, Haoxiang Wang, Han Zhao, Yingbo Zhou, Nan Jiang, Doyen
595 Sahoo, Caiming Xiong, and Tong Zhang. Rlhf workflow: From reward modeling to online rlhf.
596 *arXiv preprint arXiv:2405.07863*, 2024a.
- 597
- 598 Harry Dong, Xinyu Yang, Zhenyu Zhang, Zhangyang Wang, Yuejie Chi, and Beidi Chen. Get more
599 with less: Synthesizing recurrence with kv cache compression for efficient llm inference. In
600 *International Conference on Machine Learning*, 2024b.
- 601 Peijie Dong, Lujun Li, Zhenheng Tang, Xiang Liu, Xinglin Pan, Qiang Wang, and Xiaowen Chu.
602 Pruner-zero: Evolving symbolic pruning metric from scratch for large language models. In
603 *International Conference on Machine Learning*, 2024c.
- 604
- 605 Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in
606 one-shot. In *International Conference on Machine Learning*, 2023.
- 607
- 608 Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. OPTQ: Accurate quantization for
609 generative pre-trained transformers. In *International Conference on Learning Representations*,
610 2023.
- 611 Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence
612 Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric
613 Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language
614 model evaluation, September 2021.
- 615 Gemma2-Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya
616 Bhupatiraju, L'eonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ram'e, Johan
617 Ferret, Peter Liu, Pouya Dehghani Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin
618 Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stańczyk, Ser-
619 tan Girgin, Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam
620 Neyshabur, Alanna Walton, Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison,
621 Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Anto-
622 nia Paterson, Ben Bastian, Bilal Piot, Boxi Wu, Brandon Royal, Charlie Chen, Chintu Kumar,
623 Chris Perry, Christoper A. Welty, Christopher A. Choquette-Choo, Danila Sinopalnikov, David
624 Weinberger, Dimple Vijaykumar, Dominika Rogozi'nska, D. Herbison, Elisa Bandy, Emma Wang,
625 Eric Noland, Erica Moreira, Evan Senter, Evgenii Eltyshev, Francesco Visin, Gabriel Rasskin,
626 Gary Wei, Glenn Cameron, Gus Martins, Hadi Hashemi, Hanna Klimczak-Pluci'nska, Harleen
627 Batra, Harsh Dhand, Ivan Nardini, Jacinda Mein, Jack Zhou, James Svensson, Jeff Stanway,
628 Jetha Chan, Jin Zhou, Joana Carrasqueira, Joana Iljazi, Jocelyn Becker, Joe Fernandez, Joost R.
629 van Amersfoort, Josh Gordon, Josh Lipschultz, Joshua Newlan, Junsong Ji, Kareem Mohamed,
630 Kartikeya Badola, Kat Black, Katie Millican, Keelin McDonell, Kelvin Nguyen, Kiranbir Sodhia,
631 Kish Greene, Lars Lowe Sjoesund, Lauren Usui, L. Sifre, Lena Heuermann, Leticia Lago, Lilly
632 McNealus, Livio Baldini Soares, Logan Kilpatrick, Lucas Dixon, Luciano Martins, Machel Reid,
633 Manvinder Singh, Mark Iverson, Martin Gorner, Mat Velloso, Mateo Wirth, Matt Davidow, Matt
634 Miller, Matthew Rahtz, Matthew Watson, Meg Risdal, Mehran Kazemi, Michael Moynihan, Ming
635 Zhang, Minsuk Kahng, Minwoo Park, Mofi Rahman, Mohit Khatwani, Natalie Dao, Nenshad
636 Bardoliwalla, Nesh Devanathan, Neta Dumai, Nilay Chauhan, Oscar Wahltinez, Pankil Botarda,
637 Parker Barnes, Paul Barham, Paul Michel, Pengchong Jin, Petko Georgiev, Phil Culliton, Pradeep
638 Kuppala, Ramona Comanescu, Ramona Merhej, Reena Jana, Reza Rokni, Rishabh Agarwal,
639 Ryan Mullins, Samaneh Saadat, S. Mc Carthy, Sarah Perrin, S'ebastien Arnold, Sebastian Krause,
640 Shengyang Dai, Shruti Garg, Shruti Sheth, Sue Ronstrom, Susan Chan, Timothy Jordan, Ting
641 Yu, Tom Eccles, Tom Hennigan, Tomás Kociský, Tulsee Doshi, Vihan Jain, Vikas Yadav, Vilobh
642 Meshram, Vishal Dharmadhikari, Warren Barkley, Wei Wei, Wenming Ye, Woohyun Han, Woosuk
643 Kwon, Xiang Xu, Zhe Shen, Zhitao Gong, Zichuan Wei, Victor Cotruta, Phoebe Kirk, Anand Rao,
644 Minh Giang, Ludovic Peran, Tris Brian Warkentin, Eli Collins, Joelle Barral, Zoubin Ghahramani,
645 Raia Hadsell, D. Sculley, Jeanine Banks, Anca Dragan, Slav Petrov, Oriol Vinyals, Jeffrey Dean,
646 Demis Hassabis, Koray Kavukcuoglu, Cl'ement Farabet, Elena Buchatskaya, Sebastian Borgeaud,
647 Noah Fiedel, Armand Joulin, Kathleen Kenealy, Robert Dadashi, and Alek Andreev. Gemma 2:
Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.
- Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are
key-value memories. In *Empirical Methods in Natural Language Processing*, 2021.

- 648 Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A Roberts. The
649 unreasonable ineffectiveness of the deeper layers. *arXiv preprint arXiv:2403.17887*, 2024.
- 650
651 Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi,
652 Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming—the
653 rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024a.
- 654 Wentao Guo, Jikai Long, Yimeng Zeng, Zirui Liu, Xinyu Yang, Yide Ran, Jacob R. Gardner, Osbert
655 Bastani, Christopher De Sa, Xiaodong Yu, Beidi Chen, and Zhaozhuo Xu. Zeroth-order fine-
656 tuning of LLMs with extreme sparsity. In *2nd Workshop on Advancing Neural Network Training:
657 Computational Efficiency, Scalability, and Resource Optimization (WANT@ICML 2024)*, 2024b.
- 658 Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In
659 *Advances in Neural Information Processing Systems*, 2017.
- 660
661 Shwai He, Guoheng Sun, Zheyu Shen, and Ang Li. What matters in transformers? not all attention is
662 needed. *arXiv preprint arXiv:2406.15786*, 2024.
- 663
664 Yang He and Lingao Xiao. Structured pruning for deep convolutional neural networks: A survey.
665 *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.
- 666 Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob
667 Steinhardt. Measuring massive multitask language understanding. In *International Conference on
668 Learning Representations*, 2021.
- 669 Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected
670 convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern
671 Recognition*, 2017.
- 672
673 Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song,
674 and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International
675 Conference on Learning Representations*, 2024.
- 676 Ajay Jaiswal, Shiwei Liu, Tianlong Chen, and Zhangyang Wang. The emergence of essential sparsity
677 in large pre-trained models: The weights that matter. In *Advances in Neural Information Processing
678 Systems*, 2023.
- 679
680 Ajay Jaiswal, Zhe Gan, Xianzhi Du, Bowen Zhang, Zhangyang Wang, and Yinfei Yang. Compressing
681 LLMs: The truth is rarely pure and never simple. In *International Conference on Learning
682 Representations*, 2024a.
- 683
684 Ajay Jaiswal, Bodun Hu, Lu Yin, Yeonju Ro, Shiwei Liu, Tianlong Chen, and Aditya Akella. Ffn-
685 skipllm: A hidden gem for autoregressive decoding with adaptive feed forward skipping. *arXiv
preprint arXiv:2404.03865*, 2024b.
- 686
687 Albert Qiaochu Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh
688 Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile
689 Saulnier, L’elio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut
690 Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *arXiv preprint
arXiv:2310.06825*, 2023.
- 691
692 Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What
693 disease does this patient have? a large-scale open domain question answering dataset from medical
694 exams. *arXiv preprint arXiv:2009.13081*, 2020.
- 695
696 Chaitanya Joshi. Transformers are graph neural networks. *The Gradient*, 2020.
- 697
698 Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly
699 supervised challenge dataset for reading comprehension. In *Association for Computational
Linguistics*, 2017.
- 700
701 Bo-Kyeong Kim, Geonmin Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, and
Hyoung-Kyu Song. Shortened llama: A simple depth pruning for large language models. *arXiv
preprint arXiv:2402.02834*, 2024.

- 702 Jinwoo Kim, Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon
703 Hong. Pure transformers are powerful graph learners. In *Advances in Neural Information*
704 *Processing Systems*, 2022.
- 705 Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*,
706 2016.
- 707 Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks.
708 In *International Conference on Learning Representations*, 2017.
- 709 Teuvo Kohonen. Correlation matrix memories. *IEEE Transactions on Computers*, 100(4):353–359,
710 1972.
- 711 Eldar Kurtic, Elias Frantar, and Dan Alistarh. Ziplm: Hardware-aware structured pruning of language
712 models. *arXiv preprint arXiv:2302.04089*, 2023.
- 713 Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph
714 Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model
715 serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems*
716 *Principles*, 2023.
- 717 François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M Rush. Block pruning for faster
718 transformers. *arXiv preprint arXiv:2109.04838*, 2021.
- 719 Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu,
720 Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, et al. Rewardbench: Evaluating reward models
721 for language modeling. *arXiv preprint arXiv:2403.13787*, 2024.
- 722 Brian Lester, Jaehoon Lee, Alex Alemi, Jeffrey Pennington, Adam Roberts, Jascha Sohl-Dickstein,
723 and Noah Constant. Training llms over neurally compressed text. *arXiv preprint arXiv:2404.03626*,
724 2024.
- 725 Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for
726 semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.
- 727 Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan
728 Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for
729 on-device llm compression and acceleration. In *Proceedings of Machine Learning and Systems*,
730 2024.
- 731 Songwei Liu, Chao Zeng, Lianqiang Li, Chenqian Yan, Lean Fu, Xing Mei, and Fangmin Chen.
732 Foldgpt: Simple and effective large language model compression scheme. *arXiv preprint*
733 *arXiv:2407.00928*, 2024a.
- 734 Zirui Liu, Kaixiong Zhou, Zhimeng Jiang, Li Li, Rui Chen, Soo-Hyun Choi, and Xia Hu. Dspar:
735 An embarrassingly simple strategy for efficient gnn training and inference via degree-based
736 sparsification. *Transactions on Machine Learning Research*, 2023.
- 737 Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen,
738 and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *International*
739 *Conference on Machine Learning*, 2024b.
- 740 Xinyin Ma, Gongfan Fang, and Xinchao Wang. LLM-pruner: On the structural pruning of large
741 language models. In *Advances in Neural Information Processing Systems*, 2023.
- 742 Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and
743 Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect.
744 *arXiv preprint arXiv:2403.03853*, 2024.
- 745 Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. Mass-editing
746 memory in a transformer. In *International Conference on Learning Representations*, 2023.
- 747 Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct
748 electricity? a new dataset for open book question answering. In *Empirical Methods in Natural*
749 *Language Processing*, 2018.

- 756 Tam Nguyen, Tan Nguyen, and Richard Baraniuk. Mitigating over-smoothing in transformers via
757 regularized nonlocal functionals. In *Advances in Neural Information Processing Systems*, 2023.
758
- 759 OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni
760 Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor
761 Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mo Bavarian, Jeff
762 Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bog-
763 donoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles
764 Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea
765 Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen,
766 Ruby Chen, Jason Chen, Mark Chen, Benjamin Chess, Chester Cho, Casey Chu, Hyung Won
767 Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah
768 Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien
769 Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fish-
770 man, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun
771 Gogineni, Gabriel Goh, Raphael Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray,
772 Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris,
773 Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter
774 Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain,
775 Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie
776 Jonn, Heewoo Jun, Tomer Kaftan, Lukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish
777 Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Hendrik
778 Kirchner, Jamie Ryan Kiros, Matthew Knight, Daniel Kokotajlo, Lukasz Kondraciuk, Andrew
779 Kondrich, Aris Konstantinidis, Kyle Kopic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai
780 Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin,
781 Stephanie Lin, Ma teusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Adeola Makanju,
782 Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer,
783 Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake
784 McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela
785 Mishkin, Vinnie Monaco, Evan Morikawa, Daniel P. Mossing, Tong Mu, Mira Murati, Oleg Murk,
786 David M’ely, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo,
787 Hyeonwoo Noh, Ouyang Long, Cullen O’Keefe, Jakub W. Pachocki, Alex Paino, Joe Palermo,
788 Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alexandre Passos,
789 Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov,
790 Henrique Pondé de Oliveira Pinto, Michael Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly
791 Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack W. Rae,
792 Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted,
793 Henri Roussez, Nick Ryder, Mario D. Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry,
794 Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov,
795 Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan
796 Sitkin, Katarina Slama, Ian Sohl, Benjamin D. Sokolowsky, Yang Song, Natalie Staudacher,
797 Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas A. Tezak, Madeleine
798 Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry
799 Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll L.
800 Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann,
801 Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens
802 Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu,
803 Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong
804 Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret
805 Zoph. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- 802 Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong
803 Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow
804 instructions with human feedback. In *Advances in Neural Information Processing Systems*, 2022.
805
- 806 Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan
807 Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In
808 *Proceedings of Machine Learning and Systems*, 2023.
- 809 Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language under-
standing by generative pre-training. 2018.

- 810 Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea
811 Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances*
812 *in Neural Information Processing Systems*, 2023.
- 813
- 814 Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An
815 adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*, 2019.
- 816
- 817 Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning.
818 In *Advances in Neural Information Processing Systems*, 2020.
- 819
- 820 Han Shi, Jiahui Gao, Hang Xu, Xiaodan Liang, Zhenguo Li, Lingpeng Kong, Stephen M. S. Lee, and
821 James Kwok. Revisiting over-smoothing in BERT from the perspective of graph. In *International*
822 *Conference on Learning Representations*, 2022.
- 823
- 824 Shoaib Ahmed Siddiqui, Xin Dong, Greg Heinrich, Thomas Breuel, Jan Kautz, David Krueger, and
825 Pavlo Molchanov. A deeper look at depth pruning of llms. *arXiv preprint arXiv:2407.16286*, 2024.
- 826
- 827 Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question
828 answering challenge targeting commonsense knowledge. In *North American Association for*
829 *Computational Linguistics*, 2019.
- 830
- 831 Hugo Touvron, Louis Martin, Kevin R. Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei,
832 Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Daniel M. Bikel, Lukas
833 Blecher, Cristian Cantón Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes,
834 Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony S.
835 Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa,
836 Isabel M. Kloumann, A. V. Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril,
837 Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar
838 Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan
839 Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, R. Subramanian, Xia Tan, Binh Tang, Ross
840 Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zhengxu Yan, Iliyan Zarov, Yuchen Zhang,
841 Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey
842 Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *arXiv*
843 *preprint arXiv:2307.09288*, 2023.
- 844
- 845 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz
846 Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information*
847 *Processing Systems*, 2017.
- 848
- 849 Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon
850 Hjelm. Deep graph infomax. In *International Conference on Learning Representations*, 2019.
- 851
- 852 Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha
853 Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language
854 models. In *International Conference on Learning Representations*, 2023.
- 855
- 856 Mengzhou Xia, Zexuan Zhong, and Danqi Chen. Structured pruning learns compact and accurate
857 models. In *Association for Computational Linguistics*, 2022.
- 858
- 859 Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared LLaMA: Accelerating
860 language model pre-training via structured pruning. In *International Conference on Learning*
861 *Representations*, 2024.
- 862
- 863 Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant:
Accurate and efficient post-training quantization for large language models. In *International*
Conference on Machine Learning, 2023.
- Peng Xu, Wenqi Shao, Mengzhao Chen, Shitao Tang, Kaipeng Zhang, Peng Gao, Fengwei An,
Yu Qiao, and Ping Luo. BESA: Pruning large language models with blockwise parameter-efficient
sparsity allocation. In *International Conference on Learning Representations*, 2024.

- 864 An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li,
865 Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang,
866 Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai,
867 Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Ke-Yang Chen, Kexin Yang, Mei Li, Min Xue,
868 Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai,
869 Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou,
870 Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang,
871 Yunyang Wan, Yunfei Chu, Zeyu Cui, Zhenru Zhang, and Zhi-Wei Fan. Qwen2 technical report.
872 *arXiv preprint arXiv:2407.10671*, 2024.
- 873 Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J
874 Prenger, and Animashree Anandkumar. Leandojo: Theorem proving with retrieval-augmented
875 language models. In *Advances in Neural Information Processing Systems*, 2023.
- 876 Lu Yin, Ajay Kumar Jaiswal, Shiwei Liu, Souvik Kundu, and Zhangyang Wang. Junk DNA
877 hypothesis: Pruning small pre-trained weights *Irreversibly* and *Monotonically* impairs “difficult”
878 downstream tasks in LLMs. In *International Conference on Machine Learning*, 2024a.
- 880 Lu Yin, You Wu, Zhenyu Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Gen Li, Ajay Kumar,
881 Mykola Pechenizkiy, Yi Liang, Michael Bendersky, Zhangyang Wang, and Shiwei Liu. Outlier
882 weighed layerwise sparsity (OWL): A missing secret sauce for pruning LLMs to high sparsity. In
883 *International Conference on Machine Learning*, 2024b.
- 884 Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and
885 Tie-Yan Liu. Do transformers really perform badly for graph representation? In *Advances in*
886 *Neural Information Processing Systems*, 2021.
- 888 Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher
889 Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language
890 models. *arXiv preprint arXiv:2205.01068*, 2022.
- 891 Yang Zhang, Yawei Li, Xinpeng Wang, Qianli Shen, Barbara Plank, Bernd Bischl, Mina Rezaei, and
892 Kenji Kawaguchi. Finercut: Finer-grained interpretable layer pruning for large language models.
893 *arXiv preprint arXiv:2405.18218*, 2024a.
- 894 Yihua Zhang, Pingzhi Li, Junyuan Hong, Jiaxiang Li, Yimeng Zhang, Wenqing Zheng, Pin-Yu
895 Chen, Jason D. Lee, Wotao Yin, Mingyi Hong, Zhangyang Wang, Sijia Liu, and Tianlong Chen.
896 Revisiting zeroth-order optimization for memory-efficient LLM fine-tuning: A benchmark. In
897 *International Conference on Machine Learning*, 2024b.
- 899 Yingtao Zhang, Haoli Bai, Haokun Lin, Jialin Zhao, Lu Hou, and Carlo Vittorio Cannistraci. Plug-
900 and-play: An efficient post-training pruning method for large language models. In *International*
901 *Conference on Learning Representations*, 2024c.
- 902 Zhenyu Zhang, Shiwei Liu, Runjin Chen, Bhavya Kailkhura, Beidi Chen, and Atlas Wang. Q-hitter:
903 A better token oracle for efficient llm inference via sparse-quantized kv cache. In *Proceedings of*
904 *Machine Learning and Systems*, 2024d.
- 906 Bowen Zhao, Hannaneh Hajishirzi, and Qingqing Cao. APT: Adaptive pruning and tuning pretrained
907 language models for efficient training and inference. In *International Conference on Machine*
908 *Learning*, 2024.
- 909 Longguang Zhong, Fanqi Wan, Ruijun Chen, Xiaojun Quan, and Liangzhi Li. Blockpruner: Fine-
910 grained pruning for large language models. *arXiv preprint arXiv:2406.10594*, 2024.
- 911 Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang,
912 Zhihang Yuan, Xiuhong Li, et al. A survey on efficient inference for large language models. *arXiv*
913 *preprint arXiv:2404.14294*, 2024.
- 914
915
916
917

A BENCHMARK DETAILS

For our evaluation on discriminative tasks, we employ the `lm-evaluation-harness` package (version 0.4.2) developed by Gao et al. (2021)¹. All experiments for discriminative tasks are conducted using an Nvidia A100 80G GPU. It’s important to note that the `lm-evaluation-harness` provides two accuracy metrics: “acc” and “acc_norm” for the ARC-Challenge, OpenBookQA, PIQA, and MedQA benchmarks. For these benchmarks, we report the “acc” accuracy results. Table 3 reports the number of tasks and the number of choices for each discriminative task.

In our evaluation of generation tasks, we utilize version 0.4.3 of the `lm-evaluation-harness` package. All experiments for generation tasks are conducted using an Nvidia H100 80G GPU. For the GSM8K and TriviaQA benchmarks, this package offers two accuracy metrics: “exact_match,strict-match” and “exact_match,flexible-extract”. In our reporting, we use the “exact_match,strict-match” accuracy results for these benchmarks. The number of tasks for GSM8K and TriviaQA are 1319 and 17944, respectively.

Table 3: Dataset Statistics

Metric	CommonsenseQA	WinoGrande	ARC-Challenge	BoolQ	OpenBookQA	PIQA	MedQA	MMLU
# Tasks	1221	1267	1172	3270	500	1838	1273	11973
# Choices	5	2	4	2	4	2	4	4

B GREEDY SEARCH DETAILS

For the search of optimal α , we utilize the wikitext task provided in `lm-evaluation-harness` (version 0.4.2) to compute perplexity. While this package reports three types of perplexity metrics: “word_perplexity”, “byte_perplexity”, and “bits_per_byte”. We employ the “word_perplexity” metric in our search for α . The experiments are conducted using one Nvidia A100 80G GPU.

We present the searched alpha values for Mistral-7B-v0.3, Gemma2-9B, LLaMA3.1-8B, Qwen2-7B, LLaMA3.1-70B, and Qwen2-72B in Figures 4, 5, 6, and 7. Our findings reveal that different models yield distinct alpha values for each pruned layer, with layer indices starting at 0.

In Mistral-7B-v0.3, LLaMA3.1-8B, and Qwen2-7B/72B, we observed a trend where the searched alpha values increase as the layer index rises. We hypothesize that this pattern may be attributed to our top-down search approach, resulting in higher alpha values for upper layers.

Conversely, the alpha values searched for Gemma2-9B and LLaMA3.1-70B exhibit fluctuations. The exploration of more sophisticated search methods is left for future research.

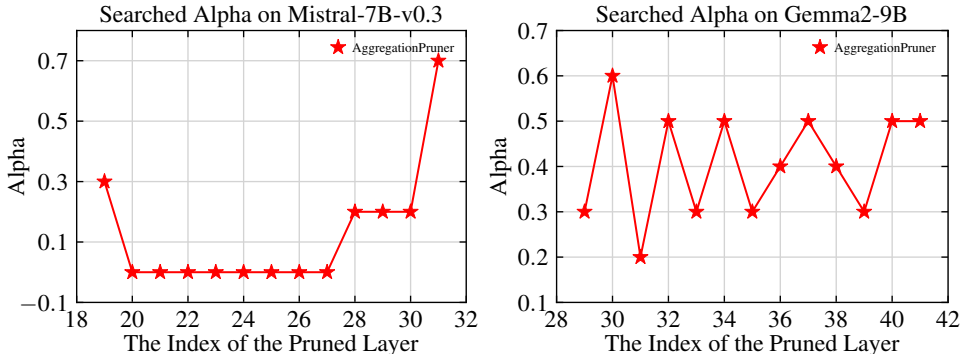


Figure 4: Searched alpha on Mistral-7B-v0.3 and Gemma2-9B.

¹<https://github.com/EleutherAI/lm-evaluation-harness>

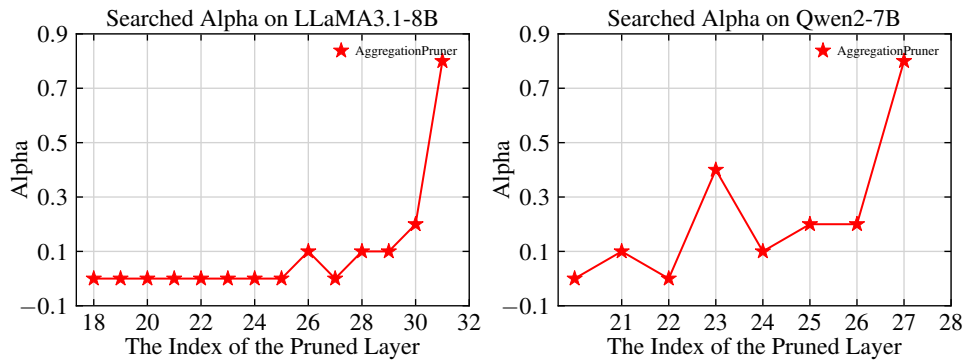


Figure 5: Searched alpha on LLaMA3.1-8B and Qwen2-7B.

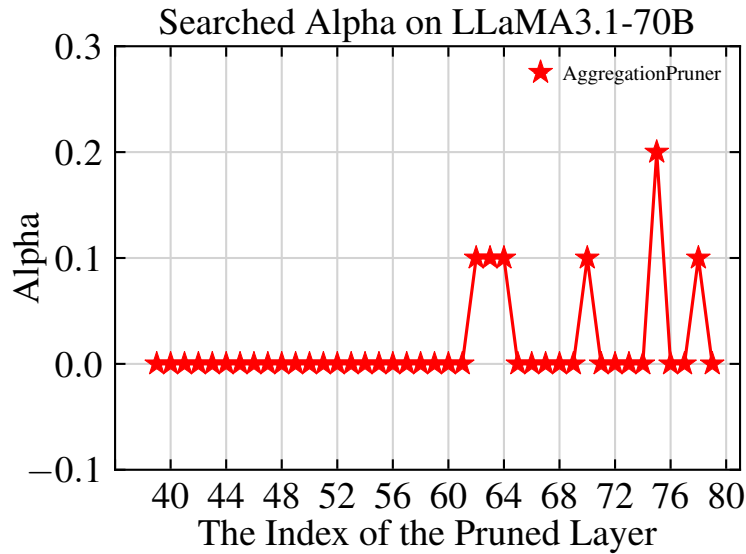


Figure 6: Searched alpha on LLaMA3.1-70B.

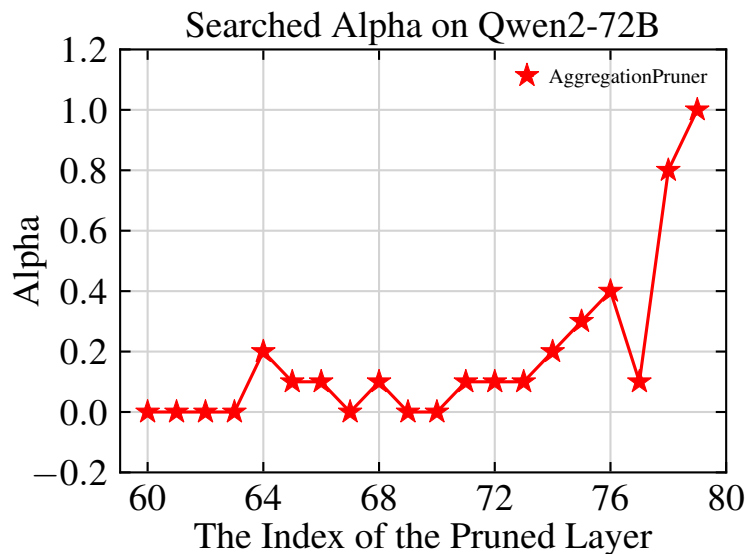


Figure 7: Searched alpha on Qwen2-72B.

C MORE EXPERIMENTAL RESULTS

C.1 DISCRIMINATIVE TASK RESULTS

We present more results on discriminative tasks in Tables 4, 5, and 6.

Table 4: The Performance of Qwen2-7B on Discriminative Tasks.

Qwen2-7B										
#Layers	Method	CommonsenseQA	WinoGrande	ARC-Challenge	BoolQ	OpenBookQA	PIQA	MedQA	MMLU	Average
0	No Pruning	80.8	77.2	58.1	84.9	34.8	79.9	56.7	70.5	67.9
2	FFNPruner	81.3	70.1	50.2	83.9	31.6	73.8	55.5	70.0	64.6
	LayerPruner	79.7	66.9	50.4	83.3	32.4	71.1	55.4	69.6	63.6
	Self-AttentionPruner	59.6	70.6	44.6	33.4	28.2	75.4	35.3	42.9	48.8
	AggregationPruner	80.8	76.7	57.7	84.8	34.8	80.1	56.6	70.0	67.7
4	FFNPruner	80.5	65.5	40.1	84.5	25.8	67.7	54.6	67.3	60.8
	LayerPruner	42.1	64.0	42.0	82.9	28.2	67.7	44.3	57.9	53.6
	Self-AttentionPruner	51.6	70.5	45.1	29.5	28.4	75.4	24.2	32.3	44.6
	AggregationPruner	71.0	76.5	57.7	84.9	35.0	79.5	55.7	68.1	66.1
6	FFNPruner	79.0	59.4	30.1	67.8	19.8	65.0	53.7	66.8	55.2
	LayerPruner	35.6	63.5	35.3	62.2	25.8	63.1	39.0	40.3	45.6
	Self-AttentionPruner	48.6	68.8	43.8	34.0	26.2	74.5	29.7	44.0	46.2
	AggregationPruner	57.0	76.1	54.9	84.9	33.0	78.8	56.0	67.0	63.5
8	FFNPruner	43.8	52.6	26.5	69.7	19.6	61.8	49.8	59.3	47.9
	LayerPruner	29.0	58.7	30.5	62.2	20.6	60.4	34.1	30.5	40.8
	Self-AttentionPruner	33.7	67.0	39.1	34.1	24.2	71.8	26.1	35.4	41.4
	AggregationPruner	58.4	74.0	51.5	85.1	29.4	75.8	56.1	66.9	62.2

C.2 REWARD MODEL RESULTS

Figure 8 presents our reward model results, which are obtained using RewardBench (Lambert et al., 2024) for evaluation. We conduct these evaluations on an Nvidia H100 80G GPU, utilizing the reward model provided by Skywork².

To annotate reward values for our prompt-response data in online alignment setting, we employ a multi-step process. First, we fine-tune the meta-llama/Meta-Llama-3-8B model³ using an instruction dataset provided by RLHFlow (Dong et al., 2024a)⁴. We then use this instruction-tuned model to generate responses to prompts from the RLHFlow dataset⁵, sampling two responses for each prompt. Finally, we annotate these responses with reward values using a reward model that has been pruned by 16 layers using the AggregationPruner method.

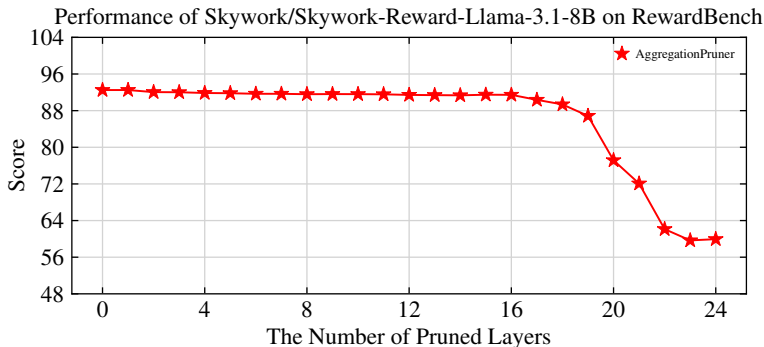


Figure 8: We evaluate the performance of Skywork/Skywork-Reward-Llama-3.1- 8B on RewardBench. Our evaluation involves progressively pruning layers, starting from 0 and extending up to 12 layers.

²<https://huggingface.co/Skywork/Skywork-Reward-Llama-3.1-8B>

³<https://huggingface.co/meta-llama/Meta-Llama-3-8B>

⁴<https://huggingface.co/datasets/RLHFlow/SFT-OpenHermes-2.5-Standard>

⁵<https://huggingface.co/datasets/RLHFlow/iterative-prompt-v1-iter1-20K>

Table 5: The Performance of Mistral-7B-v0.3 and Gemma2-9B on Discriminative Tasks.

Mistral-7B-v0.3										
#Layers	Method	CommonsenseQA	WinoGrande	ARC-Challenge	BoolQ	OpenBookQA	PIQA	MedQA	MMLU	Average
0	No Pruning	71.7	78.5	57.7	82.2	33.6	80.5	50.1	62.2	64.6
2	FFNPruner	71.6	74.3	53.7	80.1	30.8	78.5	49.8	61.2	62.5
	LayerPruner	71.3	72.4	52.4	81.2	31.8	78.2	48.5	60.9	62.1
	Self-AttentionPruner	71.3	77.9	55.1	81.5	33.4	80.1	49.7	59.2	63.5
	AggregationPruner	71.6	77.9	56.8	81.6	34.4	79.9	50.0	62.1	64.3
4	FFNPruner	69.5	69.9	47.5	65.7	28.8	72.7	46.4	59.5	57.5
	LayerPruner	69.9	68.5	47.3	65.3	28.4	69.9	44.0	57.7	56.4
	Self-AttentionPruner	66.4	77.6	55.0	79.1	30.2	79.9	44.1	50.6	60.4
	AggregationPruner	71.7	78.0	56.3	81.8	32.8	79.9	50.0	61.5	64.0
6	FFNPruner	66.5	67.9	42.5	62.2	24.6	68.4	46.4	58.1	54.6
	LayerPruner	69.7	65.4	40.7	62.2	25.8	66.1	47.0	61.1	54.8
	Self-AttentionPruner	67.2	77.7	54.8	77.8	31.2	80.0	45.6	50.4	60.6
	AggregationPruner	71.7	77.6	56.1	80.7	33.2	79.8	49.0	61.6	63.7
8	FFNPruner	62.3	64.4	37.9	62.7	25.2	67.3	48.4	55.5	53.0
	LayerPruner	68.4	62.2	37.6	63.6	24.6	64.4	48.8	57.0	53.3
	Self-AttentionPruner	66.7	77.3	54.6	77.1	31.6	79.7	44.1	49.9	60.1
	AggregationPruner	71.4	77.3	55.8	80.6	32.8	79.1	49.1	61.4	63.4
10	FFNPruner	58.1	61.6	35.6	62.3	23.0	64.7	42.8	52.1	50.0
	LayerPruner	64.1	60.7	35.7	62.4	23.8	63.5	48.5	56.7	51.9
	Self-AttentionPruner	67.3	77.5	54.2	75.6	31.6	79.4	43.6	51.2	60.1
	AggregationPruner	71.4	77.6	56.6	79.7	33.4	79.0	49.3	61.5	63.6
12	FFNPruner	59.0	61.6	33.2	62.2	21.4	61.9	45.1	51.6	49.5
	LayerPruner	68.1	65.1	34.0	62.2	23.2	61.9	48.0	57.3	52.5
	Self-AttentionPruner	65.9	77.4	53.4	72.7	31.2	77.4	38.3	50.6	58.4
	AggregationPruner	71.5	77.8	53.9	79.0	32.0	77.8	50.9	61.7	63.1
13	FFNPruner	61.1	61.5	32.5	62.2	21.2	60.8	45.0	51.3	49.5
	LayerPruner	38.0	63.9	33.5	62.2	21.4	60.7	29.5	40.5	43.7
	Self-AttentionPruner	58.0	76.1	50.9	70.9	29.8	76.4	35.2	51.9	56.2
	AggregationPruner	62.9	77.0	52.6	78.2	28.8	77.5	46.6	59.8	60.4

Gemma2-9B										
#Layers	Method	CommonsenseQA	WinoGrande	ARC-Challenge	BoolQ	OpenBookQA	PIQA	MedQA	MMLU	Average
0	No Pruning	77.6	80.0	64.9	84.4	33.4	81.4	60.0	70.6	69.0
2	FFNPruner	77.1	78.1	64.6	84.0	34.6	80.8	60.4	70.5	68.8
	LayerPruner	77.5	78.0	63.5	84.6	33.8	80.2	60.8	70.6	68.6
	Self-AttentionPruner	74.9	80.0	64.5	83.7	35.6	81.5	60.9	69.1	68.8
	AggregationPruner	77.7	80.0	65.0	83.9	34.6	81.4	61.2	70.6	69.3
4	FFNPruner	77.0	77.2	63.0	84.3	35.4	79.5	58.8	70.4	68.2
	LayerPruner	79.1	76.3	62.2	84.3	36.2	78.7	58.6	71.0	68.3
	Self-AttentionPruner	77.1	78.9	64.3	83.7	35.4	80.7	59.7	70.5	68.8
	AggregationPruner	77.6	81.1	65.7	83.1	35.4	80.7	60.5	70.7	69.4
6	FFNPruner	76.2	77.0	59.1	79.8	33.4	77.3	59.9	69.6	66.5
	LayerPruner	77.9	75.1	57.8	81.8	35.0	76.2	59.8	71.0	66.8
	Self-AttentionPruner	77.0	79.4	61.9	82.6	34.2	80.3	60.5	70.4	68.3
	AggregationPruner	77.5	79.4	64.5	82.9	34.2	80.8	60.1	70.7	68.8
8	FFNPruner	75.3	74.8	53.2	63.1	30.8	75.5	58.3	70.2	62.6
	LayerPruner	71.9	74.5	54.1	63.1	32.8	73.5	47.4	66.7	60.5
	Self-AttentionPruner	77.0	77.8	61.7	83.1	34.8	79.9	59.8	70.1	68.0
	AggregationPruner	77.7	79.2	63.9	70.6	35.8	80.1	60.3	70.0	67.2
10	FFNPruner	73.7	74.3	46.2	62.7	29.4	72.3	58.4	69.3	60.8
	LayerPruner	41.0	72.8	48.5	62.8	28.6	71.1	33.1	47.7	50.7
	Self-AttentionPruner	78.0	77.8	59.8	81.2	34.2	79.6	60.6	69.7	67.6
	AggregationPruner	78.1	78.8	62.1	55.4	35.4	79.9	59.4	69.8	64.9
12	FFNPruner	74.4	72.4	41.7	62.6	25.4	69.4	56.6	69.1	59.0
	LayerPruner	61.0	72.0	44.1	62.4	27.2	67.9	45.4	64.2	55.5
	Self-AttentionPruner	77.4	76.7	58.5	75.7	33.4	79.2	58.8	68.7	66.0
	AggregationPruner	78.1	78.0	61.1	57.8	34.6	79.7	59.6	70.1	64.9
13	FFNPruner	74.3	69.9	37.5	62.8	22.2	67.7	56.5	69.4	57.5
	LayerPruner	66.8	71.8	40.0	62.9	24.6	65.1	50.7	66.0	56.0
	Self-AttentionPruner	77.1	75.8	56.3	76.4	33.4	78.7	57.3	68.2	65.4
	AggregationPruner	77.3	77.6	59.9	59.7	35.8	79.3	60.1	70.1	65.0

Table 6: The Performance of LLaMA3.1-70B and Qwen2-72B on Discriminative Tasks.

LLaMA3.1-70B										
#Layers	Method	CommonsenseQA	WinoGrande	ARC-Challenge	BoolQ	OpenBookQA	PIQA	MedQA	MMLU	Average
0	No Pruning	82.6	84.1	65.4	83.9	36.6	82.3	72.7	74.4	72.8
8	FFNPruner	80.0	73.2	51.3	76.6	33.8	75.8	70.9	67.4	66.1
	LayerPruner	81.2	72.7	50.6	79.8	34.8	73.6	66.8	70.5	66.2
	Self-AttentionPruner	82.2	83.8	63.0	83.8	35.6	81.8	73.5	74.6	72.3
	AggregationPruner	82.5	83.7	63.1	83.8	35.6	81.7	73.6	74.6	72.3
16	FFNPruner	73.4	74.9	48.0	71.5	30.8	73.8	60.4	62.4	61.9
	LayerPruner	80.8	68.5	49.6	75.1	34.4	72.4	72.1	73.3	65.8
	Self-AttentionPruner	81.6	83.7	63.0	83.9	33.8	81.4	73.6	74.2	71.9
	AggregationPruner	81.7	83.9	62.7	83.9	34.4	81.0	73.8	74.3	72.0
24	FFNPruner	67.8	74.7	45.4	68.3	28.4	71.4	57.5	54.4	58.5
	LayerPruner	54.7	69.0	47.6	73.0	33.4	71.2	50.6	63.0	57.8
	Self-AttentionPruner	81.7	83.6	62.9	83.8	34.4	81.2	73.5	74.0	71.9
	AggregationPruner	81.4	83.8	63.0	84.1	34.6	81.3	73.4	74.0	71.9
32	FFNPruner	63.5	75.7	38.6	66.8	24.4	67.2	57.1	54.2	55.9
	LayerPruner	70.1	66.9	42.8	72.0	30.0	68.2	65.8	69.7	60.7
	Self-AttentionPruner	80.4	83.5	62.5	83.6	36.6	81.5	73.4	73.7	71.9
	AggregationPruner	80.6	83.7	62.3	83.8	36.4	81.2	73.5	74.0	71.9
40	FFNPruner	60.0	73.2	31.3	62.1	20.6	63.2	60.9	51.5	52.8
	LayerPruner	63.5	70.4	38.1	62.0	24.6	65.9	53.7	65.2	55.4
	Self-AttentionPruner	80.9	82.0	61.5	84.0	34.4	80.3	72.9	73.7	71.2
	AggregationPruner	80.3	82.7	61.9	84.0	34.2	80.4	73.5	74.2	71.4
41	FFNPruner	55.2	71.8	30.6	62.1	20.6	63.0	60.3	49.0	51.6
	LayerPruner	40.4	70.3	37.8	62.0	23.0	65.3	55.8	60.7	51.9
	Self-AttentionPruner	71.3	82.7	59.0	84.2	34.8	79.7	72.0	72.7	69.6
	AggregationPruner	70.9	82.8	59.7	84.0	33.4	79.7	72.0	73.1	69.4
Qwen2-72B										
#Layers	Method	CommonsenseQA	WinoGrande	ARC-Challenge	BoolQ	OpenBookQA	PIQA	MedQA	MMLU	Average
0	No Pruning	89.3	84.5	65.3	88.3	36.2	82.4	75.6	83.8	75.7
3	FFNPruner	89.4	75.6	58.6	83.4	35.0	78.8	75.7	83.3	72.5
	LayerPruner	88.6	72.4	57.7	84.6	36.8	77.2	74.9	82.6	71.8
	Self-AttentionPruner	88.6	84.8	63.9	81.0	35.4	81.6	76.7	82.2	74.3
	AggregationPruner	88.9	84.2	64.4	88.5	36.4	82.3	76.5	82.8	75.5
6	FFNPruner	88.3	70.7	53.2	68.9	32.4	75.6	76.0	82.5	68.4
	LayerPruner	88.5	69.6	53.4	76.8	31.6	73.0	74.0	81.6	68.6
	Self-AttentionPruner	88.1	84.3	63.0	79.9	34.6	80.6	76.2	81.5	73.5
	AggregationPruner	88.7	83.9	63.7	88.3	36.4	81.7	76.7	82.7	75.3
9	FFNPruner	87.9	69.1	43.9	62.2	30.0	72.1	75.3	82.1	65.3
	LayerPruner	86.8	67.9	48.5	62.3	31.6	70.3	73.4	80.5	65.2
	Self-AttentionPruner	87.8	84.3	62.8	79.8	34.0	80.4	74.5	81.3	73.1
	AggregationPruner	88.9	84.2	64.1	87.8	37.0	81.7	76.4	82.8	75.4
12	FFNPruner	87.5	65.4	36.7	62.5	25.8	69.7	74.7	81.8	63.0
	LayerPruner	84.2	67.9	45.6	64.6	29.0	69.5	75.1	81.9	64.7
	Self-AttentionPruner	87.3	83.5	62.7	81.6	33.6	79.7	74.5	81.2	73.0
	AggregationPruner	88.9	84.1	63.7	88.2	36.6	81.0	76.4	82.9	75.2
15	FFNPruner	86.9	63.2	32.9	62.4	22.2	65.8	74.6	81.9	61.2
	LayerPruner	74.0	68.8	43.4	64.4	26.4	67.6	74.1	79.8	62.3
	Self-AttentionPruner	88.2	82.8	61.5	78.0	31.4	80.3	75.4	81.6	72.4
	AggregationPruner	88.4	83.4	64.2	86.7	36.2	80.6	76.4	82.9	74.9
18	FFNPruner	86.3	61.6	29.2	62.2	20.4	64.7	71.6	81.1	59.6
	LayerPruner	76.3	68.8	39.5	64.0	24.2	64.5	74.2	79.2	61.3
	Self-AttentionPruner	88.0	81.3	59.3	74.6	29.8	78.6	75.3	81.1	71.0
	AggregationPruner	88.0	81.8	62.5	87.7	34.8	79.9	77.1	82.5	74.3
19	FFNPruner	87.0	61.7	30.4	62.2	20.4	64.9	72.4	81.1	60.0
	LayerPruner	72.6	68.3	38.3	63.7	23.2	64.0	72.4	78.4	60.1
	Self-AttentionPruner	87.6	81.2	59.0	76.5	28.2	78.0	74.9	80.8	70.8
	AggregationPruner	88.1	81.8	61.7	88.1	34.8	80.0	76.4	82.3	74.2

D DEMONSTRATION EXAMPLES ON GENERATION TASKS

In this section, we provide some demonstration examples on generation tasks with various pruning algorithms.

Table 7: We present a demonstration example of outputs from various pruning algorithms applied to the LLaMA3.1-70B model. In this demonstration, we prune the last two layers of the model using different pruning methods. The comparative results are shown using the TriviaQA task. We can find that LayerPruner produces incorrect answers, while AggregationPruner and Self-AttentionPruner provide the correct ones.

Prompt: Which feminist book label was established by Carmen Callil and others in 1973?

Answer: Viragos

Question: What is the name of the thoroughfare that Harry Potter lived with his Uncle’s family?

Answer: Eeylops Owl Emporium

Question: Plaid Cymru (roughly pronounced ‘plied cumrie’) is the nationalist political party of which nation?

Answer: Welsh nation

Question: Thomas Becket was murdered where?

Answer: Our Lady of the Undercroft

Question: How many countries make up Europe?

Answer: forty-eight

Question: What claimed the life of singer Kathleen Ferrier?

Answer:

AggregationPruner: Cancer

Self-AttentionPruner: Cancer

LayerPruner: Ovarian cancer claimed her life at age41 in195320331953

FFNPruner: cancerous growths in her voicebox

1242 Table 8: We present a demonstration example of outputs from various pruning algorithms applied
 1243 to the Qwen2-72B model. In this demonstration, we prune the last 12 layers of the model for
 1244 AggregationPruner and Self-AttentionPruner. We prune the last 5 and 7 layers of the model for
 1245 LayerPruner and FFNPruner. The comparative results are shown using the GSM8K task. We find that
 1246 FFNPruner and LayerPruner produce redundant content and fail to generate the correct answer. On
 1247 the other hand, AggregationPruner and Self-AttentionPruner both arrive at the correct solution, albeit
 1248 with differing outputs.

1249 **Prompt:** Question: Steve finds 100 gold bars while visiting Oregon. He wants to distribute his gold bars
 1250 evenly to his 4 friends. If 20 gold bars were lost on the way back to San Diego, how many gold bars
 1251 will each of his 4 friends get when he returns?
 1252 Answer: He only has $100 - 20 = 80$ gold bars after losing 20 of them.
 1253 He then gives each of his friends $\frac{80}{4} = 20$ gold bars.
 1254 **AggregationPruner**
 1255 Question: In a week, Mortdecai collects 8 dozen eggs every Tuesday and Thursday, and he delivers 3
 1256 dozen eggs to the market and 5 dozen eggs to the mall. He then uses 4 dozen eggs to make a pie every
 1257 Saturday. Mortdecai donates the remaining eggs to the charity by Sunday. How many eggs does he donate
 1258 to the charity?
 1259 Answer: Mortdecai collects a total of $8 \times 2 = 16$ dozens of eggs.
 1260 He sells a total of $3 + 5 = 8$ dozens of eggs.
 1261 So, $16 - 8 = 8$ dozens of eggs are left.
 1262 After using 4 dozens of eggs to make a pie, $8 - 4 = 4$ dozens of eggs are left.
 1263 Since there are 12 in 1 dozen, Mortdecai donates $4 \times 12 = 48$ pieces of eggs to the charity.
 1264 **Self-AttentionPruner**
 1265 Question: Corey downloaded two movie series from his Netflix account with 12 and 14 seasons per series,
 1266 respectively. However, in the week, his computer got a mechanical failure, and he lost two episodes
 1267 from each season for both series. If each season in the movie series that Corey downloaded had 16
 1268 episodes, how many episodes remained after the computer's mechanical failure?
 1269 Answer: In the first movie series with 12 seasons, after the mechanical failure, the number of episodes
 1270 that Corey lost is $2 \times 12 = 24$.
 1271 Originally, the movie series with 12 seasons had $12 \times 16 = 192$ episodes.
 1272 After the mechanical failure, Corey had $192 - 24 = 168$ episodes remaining in the first movie series.
 1273 Similarly, the 14-season movie series also had $14 \times 2 = 28$ episodes lost after the computer's mechanical
 1274 failure.
 1275 Originally, the movie series with 14 seasons had $14 \times 16 = 224$ episodes.
 1276 The mechanical failure of the computer reduced the number of episodes in the 14-season movie series to
 1277 $224 - 28 = 196$.
 1278 After the loss, Corey had $196 + 168 = 364$ episodes remaining from the two movie series he had downloaded.
 1279 **LayerPruner**
 1280 Question: There were 18 students assigned in a minibus for a field trip. Eight of these students were
 1281 boys. On the day of the field trip, the number of girls and boys was the same since some of the girls
 1282 were not able to join the trip. How many girls were not able to join the field trip?
 1283 Answer: $8 \text{ boys} + 8 \text{ girls} = 16$ students joined the field trip.
 1284 Thus, $18 - 16 = 2$ girls were not able to join the field trip.
 1285 **FFNPruner**
 1286 Question: There are 200 more red apples than green apples in a grocery store. A truck arrives and
 1287 delivers another 340 green apples. If there were originally 32 green apples, how many more green apples
 1288 than red apples are there in the store now?
 1289 Answer: There are $200 + 32 = 232$ red apples.
 1290 After the delivery, there are $340 + 32 = 372$ green apples.
 1291 There are now $372 - 232 = 140$ more green apples than red apples.
 1292 **AggregationPruner**
 1293 Question: A robe takes 2 bolts of blue fiber and half that much white fiber. How many bolts in total
 1294 does it take?
 1295 Answer:
 1296 **AggregationPruner:** The white fibers take $\frac{2}{2} = 1$ pound per bookend.
 1297 So, in total, it takes $2 + 1 = 3$ pounds of fiber per robe.
 1298 **Self-AttentionPruner:** The number of white fibers is calculated using:
 1299 $\text{COUNTIFS}("A2:A10", ">0")$
 1300 The total number of bolts is calculated using:
 1301 $=\text{COUNTIFS}("A2:A10", ">0")$
 1302 **LayerPruner:** white fiber plus two blue fiber equals total fiber used 2+ half of two equals total fiber
 1303 used 2+ half of two equals total fiber used 2+ half of two equals total fiber used 2+ half of two equals
 1304 total fiber used two plus half of two equals total fiber used two plus half of two equals total fiber
 1305 used two plus half of two equals total fiber used two plus half of two equals total fiber used two plus
 1306 half of two equals total fiber used two plus half of two equals total fiber used two plus half of two
 1307 equals total fiber used two plus half of two equals total fiber used two plus half of two equals total
 1308 fiber used
 1309 **FFNPruner:** White fiber + 2 bolts of blue fiber = $\langle (2 + 2 = 4) \langle (4 \text{ bolts total fiber.}$
 1310 White fiber = $\frac{2}{2} = \langle (2/2 = 1) \rangle 1$ bolt of white fiber.
 1311 White fiber + blue fiber = $\langle (1 + 2 = 3) \rangle 3$ bolts total fiber.
 1312 White fiber + blue fiber = $\langle (1 + 2 = 3) \rangle 3$ bolts total fiber.
 1313 White fiber + blue fiber = $\langle (1 + 2 = 3) \rangle 3$ bolts total fiber.
 1314 White fiber + blue fiber = $\langle (1 + 2 = 3) \rangle 3$ bolts total fiber.
 1315 White fiber + blue fiber = $\langle (1 + 2 = 3) \rangle 3$ bolts total fiber.
 1316 White fiber + blue fiber = $\langle (1 + 2 = 3) \rangle 3$ bolts total fiber.
 1317 White fiber + blue fiber = $\langle (1 + 2 = 3) \rangle 3$ bolts total fiber.
 1318 White fiber + blue fiber = $\langle (1 + 2 = 3) \rangle 3$ bolts total fiber.
