Making Large Language Models Efficient Dense Retrievers

Yibin Lei[1]*, **Shwai He**[2]*, **Ang Li**[2], **Andrew Yates**[3]
[1]University of Amsterdam [2]University of Maryland, College Park
[3]Johns Hopkins University, HLTCOE
y.lei@uva.nl, {shwaihe, angliece}@umd.edu, andrew.yates@jhu.edu

Abstract

Recent work has shown that directly fine-tuning large language models (LLMs) for dense retrieval yields strong performance, but their substantial parameter counts make them computationally inefficient. While prior studies have revealed significant layer redundancy in LLMs for generative tasks, it remains unclear whether similar redundancy exists when these models are adapted for retrieval tasks, which require encoding entire sequences into fixed representations rather than generating tokens iteratively. To this end, we conduct a comprehensive analysis of layer redundancy in LLM-based dense retrievers. We find that, in contrast to generative settings, MLP layers are substantially more prunable, while attention layers remain critical for semantic aggregation. Building on this insight, we propose EffiR, a framework for developing efficient retrievers that performs large-scale MLP compression through a coarse-to-fine strategy (coarse-grained depth reduction followed by fine-grained width reduction), combined with retrieval-specific fine-tuning. Across diverse BEIR datasets and LLM backbones, EffiR achieves substantial reductions in model size and inference cost while preserving the performance of full-size models.[1]

1 Introduction

Dense retrieval models (Karpukhin et al., 2020; Xiong et al., 2021; Hofstätter et al., 2021; Izacard et al., 2022; Ma et al., 2024) map queries and documents into a shared dense vector space, enabling efficient similarity-based search. Compared to traditional sparse methods like BM25 (Robertson et al., 1995), dense retrievers offer stronger semantic matching capabilities and have shown superior performance across a variety of information retrieval benchmarks (Bajaj et al., 2018; Thakur et al., 2021; Muennighoff et al., 2023).

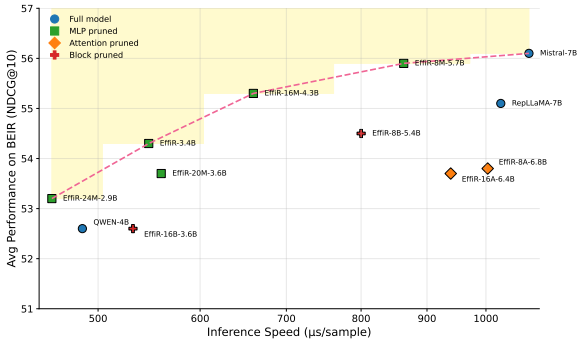


Figure 1: Effectiveness–efficiency trade-off in LLM-based dense retrievers. Each point shows a model's BEIR performance vs. inference speed. All models are fine-tuned on MS MARCO. Marker types indicate compression strategies. EffiR builds efficient models based on Mistral-7B. For example, EffiR-20M-3.6B applies our EffiR method by dropping 20 MLP layers from Mistral-7B, then fine-tuning the remaining 3.6B parameters. MLP-pruned models (green squares) consistently lie near the Pareto frontier (dashed), showing strong efficiency with minimal accuracy loss.

Large language models (LLMs) have recently emerged as powerful backbones for dense retrieval, producing high-quality text embeddings with strong generalization (Ma et al., 2024), multilingual capabilities (Li et al., 2024), and data efficiency (Luo et al., 2024). They also reduce reliance on large-scale retrieval-oriented pretraining (Wang et al., 2024) and exhibit strong instruction-following abilities (Sun et al., 2024b). However, these benefits come with a significant computational cost: these models typically rely on large models with billions of parameters, making them impractical for real-time deployment.

Recent research shows that LLMs exhibit considerable layer redundancy in *generative tasks*, where attention layers are prunable while MLP layers remain critical, enabling the removal of a substantial portion of layers with negligible degradation in performance (Gromov et al., 2024; He et al., 2024, 2025). However, despite these findings, dense

*Equal contribution
[1]Our code and models are available at https://github.com/Yibin-Lei/EffiR.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 12898–12913

retrievers are still typically built by directly fine-tuning full LLM architectures without leveraging layer redundancy (Wang et al., 2024; Ma et al., 2024), limiting their efficiency and practicality. Moreover, dense retrievers serve a fundamentally different purpose from the pretraining objective of LLMs: instead of predicting the next token iteratively, they aim to produce a single, semantically meaningful representation for the entire input sequence. This distinction motivates examining: **(i) Do the architectures of foundation models also exhibit layer redundancy in retrieval tasks? (ii) If so, how does this redundancy differ from that of generative tasks? (iii) How can this redundancy be exploited to develop more efficient retrieval models?**

To investigate these questions, we conduct a systematic analysis of layer redundancy across multiple LLM backbones, considering two settings: directly pruning off-the-shelf retrievers and pruning followed by contrastive fine-tuning. We investigate how model performance changes when different layers are dropped, following layer-dropping techniques from prior work (He et al., 2024, 2025). Interestingly, in contrast to findings in generative settings, we find that MLP layers, often viewed as repositories of factual knowledge (Zhu et al., 2020; Meng et al., 2022), are more amenable to pruning in retrieval models. Conversely, attention layers exhibit less redundancy and cannot be removed as aggressively as in generative models (He et al., 2024; Siddiqui et al., 2024), as they play a crucial role in aggregating contextual information for fine-grained semantic matching. Nonetheless, even with the higher redundancy of MLP layers, aggressive coarse-grained layer dropping alone leads to notable performance degradation beyond a certain compression ratio.

Motivated by the above findings, and given the substantial memory and computational overhead of MLP layers in retrieval scenarios, where autoregressive decoding and key-value caching are not applicable, we propose **Effi**cient **R**etriever (**EffiR**), a framework that performs large-scale retrieval-oriented MLP compression through a coarse-to-fine-grained strategy, followed by retrieval-specific fine-tuning. EffiR employs two complementary compression stages: (i) *coarse-grained depth reduction*, which removes entire low-importance MLP layers, and (ii) *fine-grained width reduction*, which adaptively compresses the intermediate dimensions of the retained MLP layers. As shown in Figure 1, MLP-pruned models (e.g., EffiR-16M-4.3B) consistently lie on or near the Pareto frontier, demonstrating that substantial MLP compression can significantly improve efficiency with minimal performance degradation for retrieval. Notably, EffiR-3.4B, which applies our full coarse-to-fine framework, achieves higher BEIR performance than EffiR-20M-3.6B, which uses only coarse-grained layer dropping, despite having a smaller parameter count. This illustrates the advantage of combining coarse-grained depth reduction with fine-grained width compression, allowing us to retain key representational capacity while reducing redundancy more effectively than coarse methods alone. Further experiments demonstrate that our coarse-to-fine framework generalizes beyond Mistral and that EffiR is competitive with widely used pruning methods while pruning only MLP layers and providing substantial speedups through structural pruning.

2 Related Works

Dense Retrievers. Early dense retrievers fine-tuned pre-trained language models (e.g., BERT (Devlin et al., 2019)) directly for retrieval tasks (Karpukhin et al., 2020; Izacard et al., 2022; Lei et al., 2023), leading to a range of methods that incorporate advanced techniques like hard negative mining (Xiong et al., 2021; Wang et al.; Hofstätter et al., 2021). More recently, LLMs have been adapted for dense retrieval (Ma et al., 2024; Weller et al., 2025; Wang et al., 2024), offering strong generalization, multilingual abilities, and less reliance on domain-specific supervision. However, these benefits come at the cost of significant computational overhead. Prior work has explored improving retrieval efficiency by reducing embedding dimensionality (Kusupati et al., 2024; Lei et al., 2025) or by accelerating similarity search using approximate nearest neighbor techniques (Kumar et al., 2024; Bruch et al., 2024), but the encoding step, often the main computational bottleneck, has received comparatively less attention. Beyond reducing embedding dimensionality alone (as in Matryoshka-style representations (Kusupati et al., 2024)), more recent 2D Matryoshka approaches train models that allow reducing both model layers and embedding dimensions, enabling further flexible effectiveness-efficiency trade-offs during encoding (Zhuang et al., 2025; LI et al., 2025).

Recently, DRAMA (Ma et al., 2025) developed small dense retrievers from LLMs through a complex three-stage data augmentation pipeline (producing over 50 million synthetic training samples) combined with pruning across all model components, including both attention and MLP layers. In contrast, we discover that pruning MLP layers alone is sufficient and use this insight to construct a simple single-stage fine-tuning strategy using only standard MSMARCO data.

Redundancy in LLMs. While increasing the depth of large language models significantly enhances their capacity (OpenAI et al., 2024; Gemini, 2024), it also introduces considerable redundancy across layers. Recent studies have addressed this by identifying and removing less critical layers. For example, Gromov et al. (2024) highlights the limited utility of deeper layers and advocates for dropping contiguous Transformer blocks. He et al. (2024) further examines the internal architecture of Transformer blocks and proposes fine-grained pruning strategies focused on attention layers, resulting in more effective layer reduction. Despite these advancements, existing work primarily focuses on generative models (Jiang et al., 2023; Grattafiori et al., 2024) that perform token-level generation. In contrast, our work focuses on embedding models (Chen et al., 2024; Wang et al.) that operate at the sequence level to produce fixed-length representations, and holistically investigates how redundancy manifests in this setting, offering new insights beyond the generative paradigm.

3 How Retrieval Models are Different

LLMs have shown promising performance in dense retrieval due to their ability to learn effective embedding representations. However, despite sharing the same underlying architectures, language modeling and dense retrieval serve distinct purposes. This section delineates these differences to motivate a distinct analysis of redundancy in dense retrievers.

Training Objectives. Dense retrievers and language models differ fundamentally in their training objectives. Language models are typically optimized with token-level predictive losses. For instance, causal language models adopt an autoregressive objective:

$$\mathcal{L}_{\text{LM}} = -\sum_{t=1}^{T} \log P(x_t \mid x_{<t}), \tag{1}$$

where the model predicts each token x_t conditioned on its leftward context $x_{<t}$. In contrast, dense retrievers are trained to produce semantically meaningful representations. A common approach is to optimize a *contrastive loss* that brings matched pairs (e.g., a query and its relevant document) closer while pushing apart mismatched pairs. One widely used formulation is the InfoNCE loss:

$$\mathcal{L} = -\log \frac{\exp(\text{sim}(q, d^+)/\tau)}{\exp(\text{sim}(q, d^+)/\tau) + \sum_{j=1}^{N} \exp(\text{sim}(q, d_j^-)/\tau)}, \tag{2}$$

where q and d^+ are the query and its corresponding positive document, d_j^- are negative samples, τ is a temperature parameter, and $\text{sim}()$ typically denotes the similarity function between extracted representations, which is usually cosine similarity or inner product. These embeddings are usually derived from the model outputs using mean pooling or by extracting the hidden state of the last token.

Divergence in Inference Pattern. The fundamental difference leads to distinct inference patterns. Specifically, generative models operate autoregressively, where local information from previously generated tokens is often sufficient to predict the next token. In contrast, dense retrieval models process the entire input sequence once and require stronger global semantic aggregation to produce fixed-length representations. This divergence suggests that attention layers, which aggregate information across tokens, and MLP layers, which perform intra-token transformations, may contribute differently in the two modeling paradigms.

On the other hand, unlike generative models, dense retrieval models obtain the representation in a single forward pass without the autoregressive decoding process, thereby eliminating the additional memory overhead (e.g., KV cache) associated with sequential attention computations. Moreover, MLP layers account for the majority of the parameters (e.g., 77.8% in Mistral-7B (Jiang et al., 2023)), and are computationally intensive due to their operations in high-dimensional spaces. This suggests focusing more heavily on the MLP layers to achieve high compression rates and promote efficiency.

Motivation. Despite the functional differences between generative and dense retrieval models, mainstream dense retrievers are still typically derived from general-purpose language models through post-finetuning, without modifying the underlying architecture. However, these architectures

E5-Mistral #Params	Full-Model 7.1B	Drop-8A 6.8B	Drop-16A 6.4B	Drop-8B 5.4B	Drop-16B 3.6B	Drop-8M 5.7B	Drop-16M 4.3B
Arguana	61.6	55.3	0.1	39.0	9.3	59.0	8.4
Climate-FEVER	37.5	32.8	2.2	1.2	3.9	39.0	9.8
DBPedia	48.6	44.8	0.6	19.0	9.7	44.7	15.2
FEVER	88.3	83.8	9.5	36.6	22.5	89.6	56.6
FiQA	57.2	51.8	2.4	9.1	8.1	53.5	14.0
HotpotQA	75.6	67.5	2.8	43.0	9.6	75.5	18.0
NFCorpus	38.7	36.1	2.6	11.4	17.5	37.4	18.2
NQ	67.1	57.0	0.4	25.6	12.1	66.1	26.0
Quora	89.3	87.1	0.6	21.5	26.8	89.1	66.0
SCIDOCS	16.8	13.0	0.1	7.3	2.2	15.2	5.1
SciFact	76.7	72.2	1.5	33.1	20.3	74.1	30.6
TREC-COVID	86.6	85.2	12.6	55.3	20.0	80.8	45.7
Touche-2020	23.6	18.8	0.0	7.2	5.6	19.1	7.6
Average	59.1	54.3	2.7	23.8	12.9	57.2	24.7

Table 1: Effectiveness (nDCG@10) and model sizes of E5-Mistral under different coarse-grained layer dropping strategies. "Full-Model" denotes the unpruned model. "Drop-kA", "Drop-kB", and "Drop-kM" indicate pruning k self-attention layers, transformer blocks, and MLP layers using our coarse-grained layer dropping method, respectively. **No recovery training is applied.**

are originally pre-trained for generative tasks and may introduce unnecessary complexity when applied to retrieval-based applications. This motivates us to examine whether general-purpose LLM architectures are overparameterized for retrieval and, if so, how their redundancy can be systematically leveraged to develop more efficient dense retrievers.

4 Are LLM-based Retrievers Redundant?

To examine redundancy in LLM-based dense retrievers, we apply the layer dropping method from He et al. (2024) that focuses on removing the whole redundant layers to improve efficiency at scale. We study this by pruning three components of LLMs: self-attention layers, MLP layers, and full transformer blocks. While prior work on generative tasks (He et al., 2024; Siddiqui et al., 2024) shows attention layers can often be pruned with little impact, it's unclear whether the same holds for embedding-based models like dense retrievers.

4.1 Redundancy Analysis via Layer Dropping

Transformer-based models are stacked by multiple transformer blocks, each containing an attention layer and an MLP layer. So a L-layer model has $2L$ sub-layers and each sub-layer has a residual connection. For our analysis, we adopt a layer dropping strategy that removes sub-layers with minimal contribution to embedding quality. This requires estimating the importance of each sub-layer. For the l-th sub-layer, we compute the importance scores:

$$S_l = M(x_l, x_{l+1}), \quad x_{l+1} = x_l + F_l(x_l), \quad (3)$$

where F_l denotes the l-th layer, and M represents the matching metric, e.g., l_2-norm and cosine similarity. To assess the importance of the l-th layer, we compare the layer's output x_{l+1} with the input x_l. If the input closely matches the full output, it indicates that the incremental contribution of $F_l(x)$ is minimal, suggesting that the layer can be removed with limited impact on performance. Following prior work showing the effectiveness of cosine similarity for identifying redundant layers (Gromov et al., 2024; He et al., 2024), we adopt it as our importance metric, i.e., $M(x, y) = 1 - \text{Cosine}(x, y)$.

Given the distinct roles of attention and MLP layers, where attention layers aggregate contextual information across tokens, while MLP layers perform intra-token transformations, we treat them as two separate groups during pruning. Specifically, we retain only the most important layers within each group. Let S_{Attn} and S_{MLP} denote the importance scores for the attention and MLP layers, respectively. The selected sets of layers are defined as:

$$\mathcal{T}_{\text{Attn}} \leftarrow \text{Argmax}(S_{\text{Attn}}, k_{\text{Attn}}), \quad (4)$$

$$\mathcal{T}_{\text{MLP}} \leftarrow \text{Argmax}(S_{\text{MLP}}, k_{\text{MLP}}), \quad (5)$$

where k_{Attn} and k_{MLP} denote the numbers of retained attention and MLP layers, respectively. The sets of selected layers are represented by $\mathcal{T}_{\text{Attn}}$ and $\mathcal{T}_{\text{MLP}}$, corresponding to the retained attention and MLP layers, respectively. The operator Argmax selects the top-k most important layers in each group.

4.2 Setup

Pruning Setup. Following He et al. (2024), we compute importance scores for each layer using 256 validation samples from the C4 corpus (Raffel et al., 2020), which is embedding and task-agnostic. We then prune the least important modules per category and evaluate the resulting models on 13 BEIR datasets (Thakur et al., 2021) using nDCG@10.

Mistral-7B #Params	Full-Model 7.1B	Drop-8A 6.8B	Drop-16A 6.4B	Drop-8B 5.4B	Drop-16B 3.6B	Drop-Last16B 3.6B	Drop-8M 5.7B	Drop-16M 4.3B	Drop-20M 3.6B	Drop-24M 2.9B	Drop-16M8A 4.0B
Arguana	58.2	56.1	47.4	56.0	44.4	43.0	56.9	54.5	51.4	45.6	53.2
Climate-FEVER	30.8	29.9	29.4	29.4	27.1	27.9	30.4	31.6	31.9	28.9	27.9
DBPedia	44.2	40.1	42.1	40.2	41.5	42.0	43.9	42.2	37.4	41.3	39.2
FEVER	83.5	79.1	77.4	81.9	76.3	76.5	82.6	83.3	79.7	80.6	79.9
FiQA	45.9	45.5	43.5	44.8	41.8	43.9	45.8	44.9	43.1	40.3	43.7
HotpotQA	70.5	68.1	66.2	67.9	65.1	65.4	71.0	70.1	67.9	67.5	67.9
NFCorpus	34.7	31.6	34.2	32.5	35.3	37.8	33.3	35.4	33.1	36.3	33.9
NQ	65.0	64.5	62.4	64.4	62.5	62.4	65.4	63.8	60.8	58.4	61.9
Quora	83.1	83.3	87.9	84.2	88.2	86.8	85.9	83.7	86.5	87.0	84.3
SCIDOCS	17.2	14.3	17.2	16.0	17.3	16.6	17.3	17.1	17.0	17.4	16.5
SciFact	75.9	73.7	76.2	74.9	74.0	73.7	76.2	76.4	75.7	74.3	75.9
TREC-COVID	84.1	83.0	86.3	82.8	82.2	85.6	83.7	84.2	85.7	83.3	85.3
Touche-2020	35.7	30.7	28.4	33.0	28.3	27.1	34.4	32.3	28.1	31.1	28.7
Average	56.1	53.8	53.7	54.5	52.6	53.0	55.9	55.3	53.7	53.2	53.7

Table 2: Effectiveness (nDCG@10) and model sizes of Mistral-7B variants **trained using MS MARCO** after coarse-grained layer dropping. We compare pruning of MLP layers (e.g., Drop-16M), attention layers (e.g., Drop-16A), full transformer blocks (e.g., Drop-16B), and their combinations (e.g., Drop-16M8A). Drop-Last16B denotes directly dropping the last 16 blocks. **Results for LLaMA3-8B, Qwen-2.5-1.5B, Qwen-2.5-3B, Qwen-2.5-7B, and ModernBERT-base are provided in Appendices A.4.**

We also report parameter counts for each variant[2]. We consider two settings: (i) directly pruning off-the-shelf retrievers and (ii) pruning base models followed by contrastive fine-tuning.

Dense Retriever Training Setup. After compressing the base model, we fine-tune it for retrieval tasks. Following prior work (Wang et al., 2024; Ma et al., 2024), we append a special <eos> token to each input text and use the hidden state of the final token as the text representation. We train all dense retrievers using the InfoNCE loss defined in Equation 2, with the temperature hyperparameter set to 0.02, and additionally apply a distillation loss. All models are trained on MSMARCO data under identical settings. Detailed training configurations are provided in Appendix A.2.

4.3 Layer Redundancy Results

4.3.1 Pruning Off-the-Shelf Retrievers

Table 1 presents the results of directly pruning the off-the-shelf E5-Mistral model without any retraining. The results reveal several notable trends about redundancy in LLM-based dense retrievers. Unlike in generation tasks, where attention layers are typically more redundant, we find that pruning attention layers leads to a drastic collapse in performance, with dropping 16 attention layers nearly zeroing out retrieval scores across several datasets. This suggests that attention remains structurally vital for producing semantically rich embeddings. However, pruning MLP layers leads to a more graceful degradation: while performance drops are still noticeable, MLP-8 and MLP-16 retain moderate effectiveness on many datasets and yield significant parameter reductions. Interestingly, MLP pruning outperforms block-level pruning despite retaining less parameters (Drop-16M vs. Drop-8B), suggesting that MLPs offer a more efficient compression axis for dense retrievers.

4.3.2 Pruning Followed by Fine-tuning

To ensure a comprehensive examination, we also investigate an alternative setup: pruning the base model first, followed by contrastive fine-tuning. The training settings are detailed in Section 4.2.

As shown in Table 2, MLP layers exhibit the most redundancy. Dropping 16 MLP layers (EffiR-16M) removes ~39% of parameters with minimal performance loss (55.3 vs. 56.1). However, more aggressive pruning (EffiR-20M, EffiR-24M) leads to sharper degradation, highlighting the limitations of depth-only compression and motivating the width-aware, coarse-to-fine strategy used in EffiR (Section 5.1).

In contrast, pruning attention layers results in minimal parameter reduction but significant performance drops (e.g., EffiR-16A drops to 53.7 nDCG@10). Block-level pruning (EffiR-8B) falls between these extremes: less efficient than targeted MLP pruning and more stable than attention pruning, but lacking the precision of module-aware compression strategies.

These trends hold consistently across various models, including LLaMA3-8B, Qwen-2.5-1.5B, Qwen-2.5-3B, Qwen-2.5-7B and ModernBERT-base (see Appendices A.4), reinforcing our design principles for EffiR: (i) prioritize MLPs as the primary compression dimension, and (ii) employ adaptive width-aware self-slimming rather than solely depth pruning to achieve superior efficiency-

[2]In the paper, we exclude the language modeling head when reporting parameter counts, since text encoding does not require it.

effectiveness trade-offs.

5 EffiR: Efficient Retriever Training

While results in Section 4 highlight the significant redundancy of MLP layers, they also indicate the limitations of depth-only pruning: removing too many layers degrades retrieval effectiveness substantially. To overcome this trade-off, we introduce EffiR (Efficient Retriever Training), a coarse-to-fine compression framework designed to explores redundancy from two complementary perspectives: **depth** and **width**. Depth reduction (i.e., layer dropping) removes the whole redundant layers to improve efficiency at scale, while width reduction adaptively compresses the remaining MLP layers to further enhance compactness while maintaining model performance.

5.1 Width Reduction via Self-Slimming

In addition to the layer depth, the width of individual layers also contributes to the overall model size. In particular, the majority of parameters arise from MLP layers, which are formulated as:

$$\mathrm{MLP}(x) = W_{\mathrm{down}}\left(\mathrm{Act}(W_{\mathrm{gate}}x) \odot W_{\mathrm{up}}x\right) + x, \quad (6)$$

where $W_{\mathrm{gate}} \in \mathbb{R}^{n\times d}$ and $W_{\mathrm{down}} \in \mathbb{R}^{d\times n}$ are the weight matrices of the MLP layer, and $\mathrm{Act}(\cdot)$ denotes an activation function. For simplicity, we omit the LayerNorm (LN) in the formulation.

The intermediate dimension n is typically much larger than the hidden size d. For instance, Mistral-7B employs an intermediate size of 14,336 in its MLP layers, more than three times its hidden size of 4096. While projecting the hidden states into higher-dimensional spaces enhances the model's representational capacity, this architectural choice substantially increases the parameter count: MLP layers alone contribute to approximately 80% of Mistral-7B's total parameters.

To remedy this problem, we further propose self-slimming for width reduction across all MLP layers. Specifically, we propose an importance indicator trainable $\mathbf{z} \in \mathbb{R}^n$ at the intermediate neurons in an MLP layer. With this indicator, an MLP layer is reformulated:

$$\mathrm{MLP}(x) = W_{\mathrm{down}}\left(\mathrm{Relu}(\mathbf{z}) \cdot \mathrm{Act}(W_{\mathrm{gate}}x) \odot (W_{\mathrm{up}}x)\right) + x. \quad (7)$$

Here, $\mathrm{ReLU}(\mathbf{z})$ serves two purposes. First, it ensures that the importance scores are non-negative, as negative values could lead to unintended cancellations. Second, it imposes a form of soft masking: a neuron with $\mathrm{ReLU}(z_i) \approx 0$ contributes minimally to the output and is a candidate for pruning. To ensure compatibility with the original MLP behavior, $\mathbf{z}$ is initialized as an all-ones vector, $\mathbb{1}^n$.

We then adopt a training-oriented approach to sparsify $\mathbf{z}$, making it trainable only during the self-slimming phase. Specifically, the overall training objective is defined as:

$$\mathcal{L} = \mathcal{L}_{\text{InfoNCE}} + \lambda \mathcal{L}_{\text{norm}}. \quad (8)$$

where $\mathcal{L}_{\text{norm}}$ is the ℓ_0-norm over $\mathrm{ReLU}(\mathbf{z})$, promoting sparsity in the active neurons, and λ is the corresponding regularization weight. The two loss terms jointly steer the scaling factors to (i) improve performance on the downstream retrieval task and (ii) remain highly sparse.

However, since the ℓ_0-norm is not differentiable and cannot be optimized directly via gradient-based methods, we use a sigmoid-based relaxation as a differentiable surrogate[3].

After only a few optimisation steps, the model learns a *slim* activation pattern in which far fewer neurons are active. We then perform global pruning by ranking all scaling factors in $\mathrm{Relu}(\mathbf{z})$ across all MLP layers and freezing the least important values to 0 while setting the remaining ones to 1. The resulting binary mask acts as a gating mechanism for neuron activation. Next, we train this sparsified model with the L_{InfoNCE} objective, under the normal dense retrieval training setting as described in Section 4.2. After training, we permanently remove all intermediate dimensions of MLPs whose corresponding scaling values are 0.

6 EffiR: Experimental Results

In this section, we present comprehensive experiments demonstrating that EffiR enhances the efficiency of LLMs for dense retrieval without compromising retrieval performance.

6.1 Experimental Setup

Model. We develop EffiR by applying our coarse-to-fine framework to the Mistral-7B-v0.1 model. In the first stage, we perform *coarse-grained MLP layer dropping*, removing 16 MLP layers identified as least important, following the pruning setup described in Section 4.2. In the second stage, we apply our fine-grained *self-slimming* method to reduce the width of the remaining MLPs by 30%. We

[3]Details on the relaxation are provided in Appendix A.1.

	RepLLaMA	Mistral-7B*	Llama-1B*	Gemma-2B*	QWEN-4B*	EffiR-8A	EffiR-16A	EffiR-16M	EffiR-20M	EffiR
#Params	6.6B	7.1B	1.2B	2.6B	3.6B	6.8B	6.4B	4.3B	3.6B	3.4B
Query-Speedup	1.05×	1.00×	6.71×	3.24×	2.22×	1.08×	1.15×	1.64×	1.93×	1.97×
Doc-Speedup	0.98×	1.00×	6.91×	3.59×	2.18×	1.08×	1.17×	1.55×	1.80×	1.82×
Arguana	48.6	58.2	49.7	57.5	48.6	56.1	47.4	54.5	51.4	52.8
Climate-FEVER	31.0	30.8	26.4	37.7	23.4	29.9	29.4	31.6	31.9	30.4
DBPedia	43.7	44.2	38.2	37.7	43.1	40.1	42.1	42.2	37.4	43.0
FEVER	83.4	83.5	81.8	78.3	80.2	79.1	77.4	83.3	79.7	82.0
FiQA	45.8	45.9	37.8	40.6	40.1	45.5	43.5	44.9	43.1	42.8
HotpotQA	68.5	70.5	65.2	65.7	65.0	68.1	66.2	70.1	67.9	67.6
NFCorpus	37.8	34.7	32.3	33.7	35.6	31.6	34.2	35.4	33.1	35.9
NQ	62.4	65.0	57.8	58.7	60.4	64.5	62.4	63.8	60.8	60.8
Quora	86.8	83.1	82.4	77.7	83.3	87.9	84.6	83.7	86.5	83.3
SCIDOCS	18.1	17.2	17.0	15.4	14.3	17.2	17.7	17.1	17.0	17.4
SciFact	75.6	75.9	70.4	72.4	73.7	76.2	72.4	76.4	75.7	75.1
TREC-COVID	84.7	84.1	81.7	76.9	83.0	86.3	84.0	84.2	85.7	83.3
Touche-2020	30.5	35.7	29.3	28.3	29.1	32.3	30.7	28.4	28.1	31.7
Average	55.1	56.1	51.5	51.5	52.6	53.8	53.7	55.3	53.7	54.3

Table 3: Retrieval performance (nDCG@10), model size, and inference speedup of EffiR and baselines on the BEIR benchmark. EffiR denotes the model trained using the full coarse-to-fine framework. EffiR-16A and EffiR-16M denote variants of EffiR that apply only coarse-grained pruning by dropping 16 attention and MLP layers, respectively, prior to training. *Mistral-7B, Llama-1B, Gemma-2B, and QWEN-4B are trained with the same retrieval supervision with no compression applied.

also examine the generalizability of EffiR under different ratios and across alternative LLM backbones in Section 7.1.

Evaluation. In line with the evaluation in Section 4, we evaluate all models on the BEIR benchmark, using nDCG@10 as the metric. To assess efficiency, we measure both the total parameter count and **inference speedup relative to the full Mistral-7B model**. Inference speedup is measured separately for query and document encoding using 1,000 randomly sampled inputs from the NQ dataset, executed on a single H100 GPU using the HuggingFace Transformers library, with `torch.compile` applied to the models.

Baselines. We compare EffiR against RepLLAMA (Ma et al., 2024), a strong LLM-based dense retriever trained on the MS MARCO dataset (Bajaj et al., 2018), the same dataset used to train our models. To isolate the effectiveness of our training framework, we also evaluate a set of small LLMs with similar release periods to Mistral-7B, trained under identical setups. These include LLaMA-3.2-1B (Grattafiori et al., 2024), Gemma-2-2B (Team et al., 2024), and Qwen-1.5-4B (Bai et al., 2023) models. All baselines are trained using the identical configurations as described in Section 4.2. In addition, we report results for layer dropping variants in which only coarse-grained layer dropping is applied prior to training.

6.2 Main Results

As shown in Table 3, EffiR achieves strong retrieval performance while significantly reducing model size and inference cost. With an average nDCG@10 of 54.3 across the BEIR benchmark, EffiR closely matches the original Mistral-7B model (56.1), despite using only ~48% of its parameters and achieving a 1.97× query-side speedup. EffiR also consistently outperforms similarly sized small LLMs such as LLaMA-1B, Gemma-2B, and Qwen-4B, though trained under the same retrieval supervision. This highlights the strength of our framework: rather than relying on compact pretrained small models with limited capacity, EffiR starts from a larger model and applies retriever-aware compression that preserves capacity in critical components while eliminating redundant computation.

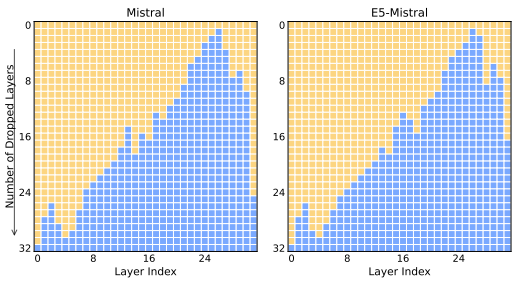
In addition to outperforming general small models, EffiR improves over intermediate coarse-grained pruned variants like EffiR-20M, suggesting that our combined approach of coarse-grained layer dropping followed by fine-grained slimming is more effective than only using the coarse-grained method. This confirms the importance of designing compression methods tailored to the retriever's architectural priorities.

7 Analysis

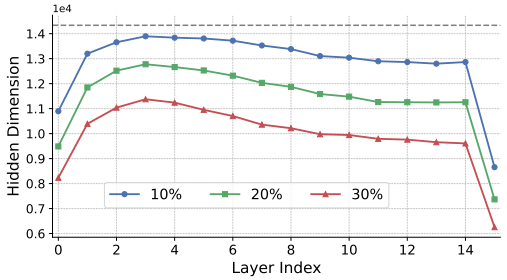
7.1 Width Reduction vs. Layer Dropping

To analyze the impact of different compression stages, we compare coarse-grained MLP layer dropping with the fine-grained self-slimming method (Figure 3), starting from the same base: dropping 16 MLP layers from the Mistral model, a strong initial trade-off.

From this point, we explore two paths: (1) dropping additional layers (EffiR-20M, EffiR-24M), and (2) applying self-slimming to progressively reduce MLP width by various sparsity ratios. The trade-off is clear: further depth pruning re-



(a) MLP Layer Dropping Heat Maps



(b) Layer-wise hidden dimension counts after width reduction under different reduction ratios.

Figure 2: Analyzing compression behavior across layers.

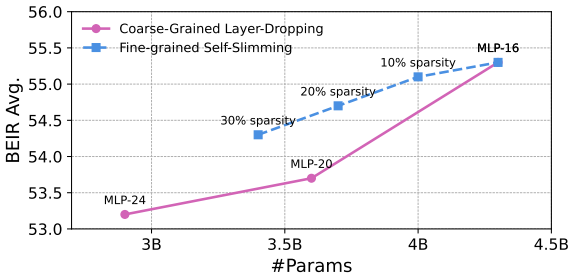


Figure 3: Comparison of coarse-grained layer dropping and fine-grained self-slimming, both starting from the same 16-layer dropped base Mistral-7B model.

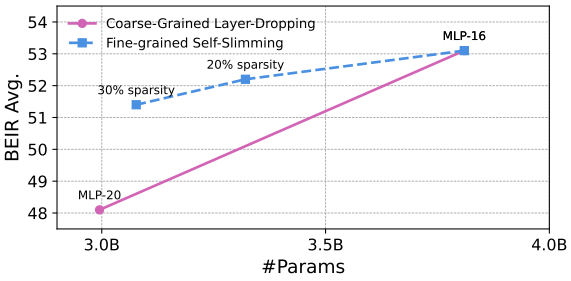


Figure 4: Comparison of coarse-grained layer dropping and fine-grained self-slimming, both starting from the same 16-layer dropped base Qwen2.5-7B model.

duces size but degrades performance, while self-slimming achieves a better efficiency-effectiveness balance. For example, 30% self-slimming outperforms EffiR-20M with fewer parameters. Results on Qwen2.5-7B (Figure 4) exhibit the same overall pattern. These results support the two-stage design of EffiR: use coarse-grained pruning to reduce redundant depth, then apply fine-grained width reduction for precise, quality-preserving compression.

7.2 Which Layers are More Redundant?

Layer Dropping. We visualize the layer-wise redundancy of MLP using dropping-order heat maps for both Mistral-7B and its retrieval-tuned variant E5-Mistral, as shown in Figure 2a. Each cell indicates whether a layer is dropped (blue) or retained (orange) during top-k pruning. Notably, the redundancy patterns remain largely consistent after retrieval fine-tuning: the E5-Mistral variant shows similar trends to the base Mistral-7B model. Later layers are generally more prunable than earlier ones, highlighting a degree of overparameterization toward the top of the model.

Self-Slimming Width-Reduction. Figure 2b presents the results of layer-wise hidden dimension counts under different overall sparsity ratios. We observe a consistent trend: deeper layers tend to be sparser than shallower layers among the remaining layers. This aligns with our layer-dropping results, where deeper layers are less important and are pruned earlier.

7.3 Comparison with Pruning Methods

Method	Sparsity	Order	BEIR Avg.
Wanda	50%	Finetune-then-Prune	49.8
		Prune-then-Finetune	54.1
SparseGPT	50%	Finetune-then-Prune	50.5
		Prune-then-Finetune	54.7
EffiR	52%	Prune-then-Finetune	54.3

Table 4: Comparison with sparsification-based pruning methods. Note the sparsity of Wanda and SparseGPT under prune-then-finetune is slightly higher than 50% since the LoRA introduces additional parameters.

We compare EffiR with Wanda (Sun et al., 2024a) and SparseGPT (Frantar and Alistarh, 2023), two widely used pruning methods that induce parameter sparsity in both attention and MLP layers. To ensure a comprehensive comparison, we evaluate them under both finetune-then-prune and prune-then-finetune settings, using the same data as EffiR. Specifically, we use their 2:4 structured pruning veriants, which prunes at the block level and can leverage optimized sparse matrix opera-

tions to achieve real acceleration, achieving $1.24\times$ speedup on LLaMA-7B as reported.

As shown in Table 4, under prune-then-finetune, both Wanda and SparseGPT achieve performance comparable to EffiR. However, their speedups rely on sparse-matrix acceleration rather than structural model changes, and the runtime improvement is not linear with sparsity (e.g., 50% sparsity yields only a $1.24\times$ speedup for LLaMA-7B, as reported in the Wanda paper). In contrast, EffiR performs hard pruning that removes entire layers and reduces hidden dimensions, which produces real reductions in computation and achieves substantially higher speedup. Moreover, sparse-matrix acceleration requires specialized kernels and hardware support and still loads the full parameter set, offering no memory savings. EffiR avoids these limitations, making it more practical and deployment-friendly.

Method	#Parameters	BEIR Avg.
LlaMA2-7B	6.6B	55.7
Sheared-LLaMA2	2.6B	49.6
EffiR-20MLP	3.9B	53.0
w/ -30%	3.4B	52.5
w/ -50%	3.1B	49.8
EffiR-24MLP	3.4B	49.7
EffiR-28MLP	2.8B	43.4

Table 5: Comparison with the structural pruning method Sheared-LLaMA on LLaMA2-7B. Dropping 20 MLP layers is denoted as -20MLP; *w/* -30% indicates reducing the width of the remaining MLP layers by 30%. EffiR-20MLP *w/* -50% achieves performance similar to Sheared-LLaMA at comparable model size, despite pruning only MLPs extremely aggressively (retaining ~19% of MLP parameters).

We further compare EffiR with another SOTA structural pruning method ShearedLlama (Xia et al., 2024), which is also used for developing the Drama model (Ma et al., 2025). For the sake of computation, we reuse the pruned LLaMA2 checkpoint released by the authors. We apply EffiR directly to the LLaMA2 model. Unlike EffiR, which prunes only MLP layers, Sheared-LLaMA prunes all parameters, giving it greater pruning flexibility.

As shown in Table 5, the two-stage design of EffiR effectively mitigates the severe performance drop that occurs when applying layer dropping alone. At comparable scale, EffiR-20MLP *w/* -50% (3.1B) achieves performance similar to Sheared-LLaMA (2.6B), despite restricting pruning to MLP layers only and using 10x less data for pruning-specific training. Notably, EffiR-20MLP *w/* -50% retains just ~19% of the original MLP parameters, directly highlighting that a large fraction of MLP capacity is redundant for retrieval.

This comparison also underscores the value of focusing on MLPs for retrieval. Sheared-LLaMA uses a pruning pipeline that compresses both attention and MLPs, yet the gap to MLP-only pruning with EffiR is small. We also emphasize that Sheared-LLaMA's pruning pipeline requires 0.4 billion training tokens from diverse domains. In contrast, EffiR achieves competitive performance using fewer tokens (~0.03 billion). Thus, even if Sheared-LLaMA is slightly smaller in total parameters, EffiR offers a simpler alternative that isolates where the redundancy lies and recovers most of the benefit at a lower pruning cost.

7.3.1 EffiR with Quantization

We apply bitsandbytes NF4 quantization (with double quantization applied) (Dettmers et al., 2023) to both the full Mistral model and the EffiR-pruned models. The results in Table 6 show that quantization behaves similarly for both the unmodified model and the EffiR-pruned model. In particular, training with EffiR does not degrade quantization performance, and in practice leads to substantial size reductions with minimal loss in effectiveness. This highlights the practical usefulness of EffiR as a technique that can be combined with quantization for even greater efficiency.

Method	Precision	Size	BEIR Avg.
Full-Mistral-Model	16bit	14.0 GB	56.1
	4bit	4.3 GB	56.0
EffiR-Mistral-8MLP	16bit	11.4 GB	55.9
	4bit	3.7 GB	55.7
EffiR-Mistral-16MLP	16bit	8.7 GB	55.3
	4bit	3.0 GB	55.3

Table 6: EffiR with quantization applied.

8 Conclusion

In this work, we investigate the redundancy of LLMs as retrievers and examine the traditional pipeline for developing dense retrievers. Specifically, we reveal the parameter redundancy inherent in directly fine-tuning base models for retrieval tasks. To address this, we propose EffiR, a two-stage framework that compresses models in depth and width, followed by fine-tuning for retrieval tasks. EffiR offers a general framework for developing retrieval models and provides valuable insights into efficient retriever design.

Limitations

We acknowledge the following limitations: (i) English-centric evaluation: EffiR is evaluated primarily on standard English retrieval benchmarks. Its effectiveness in multilingual or low-resource retrieval settings remains unexplored and may require further adaptation. (ii) Inference cost: Although EffiR enables the development of efficient retrievers that preserve performance while reducing model size, the resulting EffiR models remain slower at inference compared to smaller architectures such as BERT-base.

Acknowledgments

We acknowledge the Dutch Research Council for awarding this project access to the LUMI supercomputer, owned by the EuroHPC Joint Undertaking, hosted by CSC (Finland) and the LUMI consortium through project number NWO-2024.050.

References

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, and 1 others. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. 2018. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*.

Sebastian Bruch, Franco Maria Nardini, Cosimo Rulli, and Rossano Venturini. 2024. Efficient inverted indexes for approximate retrieval over learned sparse representations. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 152–162, New York, NY, USA. Association for Computing Machinery.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2318–2335, Bangkok, Thailand. Association for Computational Linguistics.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: massive language models can be accurately pruned in one-shot. ICML'23. JMLR.org.

Gemini. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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

Shwai He, Chaorui Deng, Ang Li, and Shen Yan. 2025. Understanding and harnessing sparsity in unified multimodal models. *arXiv preprint arXiv:2512.02351*.

Shwai He, Guoheng Sun, Zheyu Shen, and Ang Li. 2024. What matters in transformers? not all attention is needed. *arXiv preprint arXiv:2406.15786*.

Sebastian Hofstätter, Sheng-Chieh Lin, Jheng-Hong Yang, Jimmy Lin, and Allan Hanbury. 2021. Efficiently teaching an effective dense retriever with balanced topic aware sampling. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, page 113–122, New York, NY, USA. Association for Computing Machinery.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *EMNLP*.

Ramnath Kumar, Anshul Mittal, Nilesh Gupta, Aditya Kusupati, Inderjit S Dhillon, and Prateek Jain. 2024. EHI: End-to-end learning of hierarchical index for

efficient dense retrieval. *Transactions on Machine Learning Research*.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, and Ali Farhadi. 2024. Matryoshka representation learning. *arXiv preprint arXiv:2205.13147*.

Yibin Lei, Liang Ding, Yu Cao, Changtong Zan, Andrew Yates, and Dacheng Tao. 2023. Unsupervised dense retrieval with relevance-aware contrastive pretraining. In *Findings of the Association for Computational Linguistics: ACL 2023*. Association for Computational Linguistics.

Yibin Lei, Tao Shen, Yu Cao, and Andrew Yates. 2025. Enhancing lexicon-based text embeddings with large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 18986–19001, Vienna, Austria. Association for Computational Linguistics.

Chaofan Li, MingHao Qin, Shitao Xiao, Jianlyu Chen, Kun Luo, Yingxia Shao, Defu Lian, and Zheng Liu. 2024. Making text embedders few-shot learners. *arXiv preprint arXiv:2409.15700*.

Xianming LI, Zongxi Li, Jing Li, Haoran Xie, and Qing Li. 2025. ESE: Espresso sentence embeddings. In *The Thirteenth International Conference on Learning Representations*.

Kun Luo, Minghao Qin, Zheng Liu, Shitao Xiao, Jun Zhao, and Kang Liu. 2024. Large language models as foundations for next-gen dense retrieval: A comprehensive empirical assessment. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1354–1365, Miami, Florida, USA. Association for Computational Linguistics.

Xueguang Ma, Xi Victoria Lin, Barlas Oguz, Jimmy Lin, Wen-tau Yih, and Xilun Chen. 2025. DRAMA: Diverse augmentation from large language models to smaller dense retrievers. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 30170–30186, Vienna, Austria. Association for Computational Linguistics.

Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. 2024. Fine-tuning llama for multi-stage text retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 2421–2425, New York, NY, USA. Association for Computing Machinery.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in gpt. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2023. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1).

Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, and 1 others. 1995. Okapi at trec-3. *Nist Special Publication Sp*, 109:109.

Shoaib Ahmed Siddiqui, Xin Dong, Greg Heinrich, Thomas Breuel, Jan Kautz, David Krueger, and Pavlo Molchanov. 2024. A deeper look at depth pruning of LLMs. In *ICML 2024 Workshop on Theoretical Foundations of Foundation Models*.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2024a. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*.

Weiwei Sun, Zhengliang Shi, Wu Jiu Long, Lingyong Yan, Xinyu Ma, Yiding Liu, Min Cao, Dawei Yin, and Zhaochun Ren. 2024b. MAIR: A massive benchmark for evaluating instructed retrieval. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14044–14067, Miami, Florida, USA. Association for Computational Linguistics.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, and 1 others. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Nandan Thakur, Nils Reimers, Andreas Sanii, and Iryna Gurevych. 2021. Beir: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *NeurIPS*.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024. Improving text embeddings with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11897–11916, Bangkok, Thailand. Association for Computational Linguistics.

Orion Weller, Benjamin Van Durme, Dawn Lawrie, Ashwin Paranjape, Yuhao Zhang, and Jack Hessel. 2025. Promptriever: Instruction-trained retrievers can be prompted like language models. In *The Thirteenth International Conference on Learning Representations*.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2024. Sheared LLaMA: Accelerating language model pre-training via structured pruning. In *The Twelfth International Conference on Learning Representations*.

Lee Xiong, Chenyan Wu, Ye Xiong, Jian Luan, Keith Rogriguez, Luke Zettlemoyer, and Mingyang Sun. 2021. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *ICLR*.

Chen Zhu, Ankit Singh Rawat, Manzil Zaheer, Srinadh Bhojanapalli, Daliang Li, Felix Yu, and Sanjiv Kumar. 2020. Modifying memories in transformer models. *arXiv preprint arXiv:2012.00363*.

Shengyao Zhuang, Shuai Wang, Fabio Zheng, Bevan Koopman, and Guido Zuccon. 2025. Starbucks-v2: Improved training for 2d matryoshka embeddings. *arXiv preprint arXiv:2410.13230*.

A Appendix

A.1 Sigmoid Surrogate for ℓ_0-norm

To encourage sparsity during training without enforcing hard thresholding, we adopt a simple and efficient sigmoid-based surrogate inspired by the ℓ_0-norm. Specifically, for a set of parameters $\{x_i\}$, the ℓ_0-norm counts the number of non-zero entries:

$$\|\mathbf{x}\|_0 = \sum_{i=1}^{d} \mathbb{I}[x_i \neq 0].$$

This objective is discontinuous and therefore incompatible with standard gradient-based optimization. We therefore use a differentiable surrogate based on the sigmoid of the parameter magnitude:

$$\tilde{\mathcal{R}}(\mathbf{x}) = \sum_{i=1}^{d} \sigma(\beta|x_i|),$$

where $\sigma(\cdot)$ denotes the sigmoid function and $\beta > 0$ controls the sharpness of the transition. Note that $\sigma(0) = 0.5$, so this surrogate is not a direct approximation of $\mathbb{I}[x_i \neq 0]$; rather, when minimized, it softly encourages parameters to shrink toward zero by assigning the lowest penalty to zero-valued entries and increasingly larger penalties to non-zero magnitudes. Increasing β sharpens the transition around the origin (i.e., it more strongly separates small from large magnitudes), but overly large values may harm gradient stability; in practice, we fix $\beta = 5.0$.

A.2 Training Configurations

We train each model for one epoch using LoRA, applied to the `v_proj`, `q_proj`, `k_proj`, `gate_proj`, `down_proj`, `o_proj`, `up_proj` layers, with a rank of 32 and $\alpha = 64$. The learning rate is set to 1e-4, and each sample consists of one positive and seven negatives. We also include in-batch negatives and apply the KL-divergence loss to distill ranking scores from the BGE-reranker. For the self-slimming setup, we fine-tune the scaling factors for 500 steps using full-parameter training and apply a regularization weight λ of 1e-8. Afterward, we prune 30% of the intermediate dimensions in the MLP layers based on their learned scaling factors, removing the dimensions with the lowest values across all layers.

A.3 BEIR Statisitcs

We provide the detailed BEIR statistics in Table 7.

Dataset	#Test	#Corpus
Scifact	300	5,183
Arguana	1,406	8,674
Trec-Covid	50	171,332
FiQA-2018	648	57,638
DBPedia	400	4,635,922
NFCorpus	323	3,633
NQ	3,452	2,681,468
HotpotQA	7,405	5,233,329
Touche-2020	49	382,545
Quora	10,000	522,931
SCIDOCS	1,000	25,657
FEVER	6,666	5,416,568
Climate-FEVER	1,535	5,416,593

Table 7: Dataset Statistics

A.4 Additional Layer Dropping Results

We present the results of coarse-grained layer dropping analysis on the LLaMA3-8B, Qwen-2.5-1.5B, Qwen-2.5-3B, Qwen-2.5-7B, and ModernBERT-base in Tables 8, 9, 10, 11, and 12, with contrastive learning applied. Similar to Mistral-7B, dropping MLP layers results in a smaller performance degradation compared to pruning attention layers or entire transformer blocks across these models.

A.5 Results on Retrieval Latency

We find that different EffiR variants and the original model, which have the same embedding dimension, achieve comparable retrieval latency: ~2.28 ms/query on the NFCorpus dataset using the Faiss Flat index on a single core of the AMD EPYC 7763 CPU. This indicates that EffiR does not introduce any retrieval-side overhead.

A.6 Results on Embedding Space Alignment

We compute the embedding-space isotropy values (based on average pairwise cosine similarity) for both the original full model and EffiR-20MLP *w/* -20% using 10,000 MS MARCO queries. The two models obtain comparable isotropy scores (0.28 vs. 0.34), suggesting that EffiR largely preserves the geometric structure of the embedding space.

A.7 Dropping-Order Heat Maps

Figure 5 presents drop-order heat maps visualizing the layer-wise redundancy of attention layers and Transformer blocks in both Mistral-7B and its retrieval-tuned variant, E5-Mistral. As with MLP layers, the later layers tend to be more redundant and thus more prunable.

A.8 Why Attention is Critical: a Case Study

We have shown that attention layers are more critical in embedding models than in generative models. To further explore this, we examine how selectively dropping MLP or attention layers impacts retrieval performance.

As shown in Figure 6, models based on last-token embeddings (e.g., E5-Mistral (Wang et al., 2024)) lose the ability to distinguish between positive and negative documents when attention layers are dropped, whereas those with reduced MLP layers largely retain this capability. We hypothesize this is because removing attention layers disrupts the aggregation of contextual information, leading to degraded sequence representations. In contrast, pruning MLP layers preserves the information flow from context tokens to the final token, thereby maintaining retrieval effectiveness.

On the other hand, as shown in Figure 7, models using mean-pooling embeddings demonstrate greater robustness to the removal of attention layers compared to those using last-token-based embeddings. This is likely because the mean-pooled embedding still aggregates information from all tokens in the sequence, effectively preserving contextual information.

LLAMA3-8B	Full-Model	EffiR-8A	EffiR-16A	EffiR-8B	EffiR-16B	EffiR-8M	EffiR-16M	EffiR-20M	EffiR-24M	EffiR-16M8A
Arguana	48.6	46.5	40.2	50.4	37.5	51.5	52.9	42.1	42.1	44.5
Climate-FEVER	31.0	29.1	26.5	30.5	27.3	29.4	28.9	29.6	21.1	28.8
DBPedia	43.7	43.0	35.0	38.9	34.2	43.9	43.1	43.8	33.6	39.1
FEVER	83.4	81.7	79.1	80.9	78.4	83.2	83.3	84.5	68.1	81.2
FiQA	45.8	41.9	34.9	40.6	32.1	42.4	41.4	39.1	31.5	39.1
HotpotQA	68.5	67.5	58.5	66.9	61.2	70.5	68.6	67.3	57.5	65.3
NFCorpus	37.8	33.8	32.0	31.1	28.1	35.4	34.9	35.5	30.9	35.1
NQ	62.4	63.9	58.1	62.1	56.5	63.8	62.4	61.7	49.4	61.0
Quora	86.8	86.5	85.9	87.9	85.9	87.0	87.1	86.7	83.7	86.8
SCIDOCS	18.1	15.8	12.7	14.8	12.3	17.6	15.8	16.4	15.3	15.8
SciFact	75.6	73.3	69.0	73.2	69.0	74.5	72.8	73.5	64.7	73.8
TREC-COVID	84.7	81.1	78.9	79.7	76.3	82.1	83.1	81.2	75.5	82.9
Touche-2020	30.5	30.3	25.4	29.5	24.4	32.6	31.4	31.6	26.3	26.6
Average	55.5	53.4	48.9	52.8	47.9	54.9	54.3	53.3	46.1	52.3

Table 8: Effectiveness (nDCG@10) of LLAMA3-8B variants trained after coarse-grained layer dropping across different architectural components. We compare pruning of MLP layers (e.g., EffiR-16M), attention layers (e.g., EffiR-16A), full transformer blocks (e.g., EffiR-16B), and their combinations (e.g., EffiR-16M8A).

Qwen2.5-1.5B	Full-Model	EffiR-8A	EffiR-16A	EffiR-8M	EffiR-16M	EffiR-20M
Arguana	53.8	50.5	41.5	52.0	49.1	43.9
Climate-FEVER	26.6	25.5	20.3	25.7	25.2	21.2
DBPedia	42.0	38.4	31.3	40.6	38.2	32.4
FEVER	78.3	73.0	66.2	79.6	77.9	73.7
FiQA	38.5	35.0	26.2	37.3	34.0	30.2
HotpotQA	64.1	59.8	41.8	63.3	60.5	55.8
NFCorpus	35.7	36.1	32.1	35.6	35.8	30.1
NQ	57.8	55.1	44.8	55.8	53.6	46.2
Quora	86.4	86.9	83.6	86.8	84.8	82.6
SCIDOCS	17.8	16.9	12.9	17.5	15.4	13.3
SciFact	73.0	69.3	56.5	70.6	69.3	64.8
TREC-COVID	82.8	82.7	68.7	83.2	83.6	77.9
Touche-2020	30.7	26.3	24.6	28.9	29.8	29.9
Average	52.9	50.4	42.3	52.1	50.6	46.3

Table 9: Effectiveness (nDCG@10) of Qwen2.5-1.5B variants trained after coarse-grained layer dropping across different architectural components.

Qwen2.5-3B	Full-Model	EffiR-8A	EffiR-16A	EffiR-8M	EffiR-16M	EffiR-20M
Arguana	53.6	46.5	38.3	51.1	45.6	43.7
Climate-FEVER	25.2	20.9	19.3	24.3	20.3	21.4
DBPedia	42.5	37.2	31.2	40.5	38.5	37.5
FEVER	82.2	74.0	71.7	78.5	77.2	75.0
FiQA	41.8	37.4	26.6	40.5	36.8	35.8
HotpotQA	66.8	56.9	43.8	63.8	61.6	59.9
NFCorpus	36.5	35.6	32.9	35.7	33.7	32.5
NQ	61.1	58.3	50.4	59.1	54.0	52.7
Quora	88.2	86.5	83.6	87.1	85.0	84.5
SCIDOCS	16.5	14.5	11.1	16.8	16.1	15.8
SciFact	74.0	70.3	57.1	74.5	70.8	69.7
TREC-COVID	84.4	81.9	75.1	85.5	82.5	82.6
Touche-2020	34.0	28.7	29.5	32.9	31.5	30.3
Average	54.4	49.9	43.9	53.1	50.3	49.3

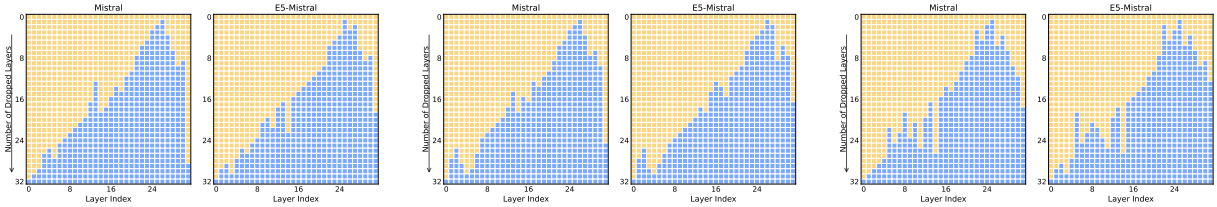
Table 10: Effectiveness (nDCG@10) of Qwen2.5-3B variants trained after coarse-grained layer dropping across different architectural components.

Qwen2.5-7B	Full-Model	EffiR-8A	EffiR-16A	EffiR-8M	EffiR-16M	EffiR-20M
Arguana	53.1	50.0	43.6	51.9	48.5	45.3
Climate-FEVER	24.2	25.0	22.8	26.7	25.6	21.4
DBPedia	44.2	44.3	42.0	44.0	42.3	36.1
FEVER	81.8	70.3	78.2	82.3	81.2	76.4
FiQA	43.9	37.7	36.6	40.0	38.4	31.3
HotpotQA	67.9	65.5	64.3	67.8	65.6	60.0
NFCorpus	37.1	35.7	34.6	37.4	36.5	32.4
NQ	62.8	60.1	58.2	61.6	57.9	51.1
Quora	88.6	87.7	86.8	87.6	87.2	78.7
SCIDOCS	18.1	17.1	15.8	17.8	18.1	15.5
SciFact	74.1	71.9	69.9	74.0	73.2	69.4
TREC-COVID	83.7	81.5	86.2	83.8	84.0	77.8
Touche-2020	32.7	25.2	28.5	32.9	31.2	33.2
Average	54.8	51.7	51.3	54.4	53.1	48.4

Table 11: Effectiveness (nDCG@10) of Qwen2.5-7B variants trained after coarse-grained layer dropping across different architectural components.

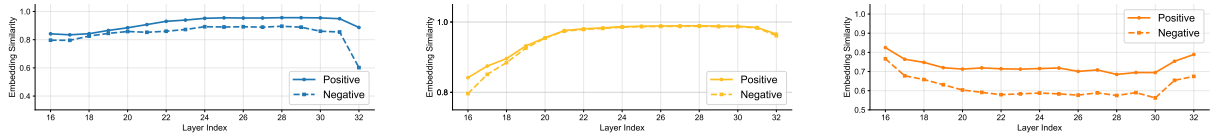
ModernBERT-base	Full-Model	EffiR-8A	EffiR-16A	EffiR-8M	EffiR-16M	EffiR-20M
Arguana	51.6	44.7	0.1	41.9	37.0	35.6
Climate-FEVER	20.1	8.7	0.0	17.6	19.3	12.5
DBPedia	26.0	12.0	0.0	17.9	14.4	7.6
FEVER	66.4	52.0	0.0	52.3	52.4	32.6
FiQA	28.5	11.5	0.2	21.4	17.5	15.1
HotpotQA	47.7	23.7	0.0	44.4	36.0	32.4
NFCorpus	24.5	22.3	0.6	21.5	19.1	15.9
NQ	44.2	35.7	0.0	35.5	27.9	23.4
Quora	79.9	69.6	0.0	83.7	79.3	77.0
SCIDOCS	12.4	2.3	0.0	9.6	7.8	4.7
SciFact	58.6	48.0	0.0	54.6	48.8	40.1
TREC-COVID	77.8	67.1	0.0	72.5	65.4	60.7
Touche-2020	26.7	25.8	0.0	24.7	23.4	25.1
Average	43.4	32.6	0.1	38.3	34.5	29.4

Table 12: Effectiveness (nDCG@10) of ModernBERT-base variants trained after coarse-grained layer dropping across different architectural components.



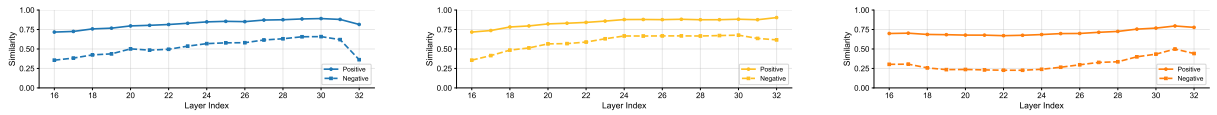
(a) Block (b) MLP (c) Attention

Figure 5: **Dropping-order heat maps** for the base model (Mistral-7B) and its embedding variant (E5-Mistral).



(a) Baseline (b) Attention Drop (c) MLP Drop

Figure 6: Layer-wise query–document similarity for last-token-based retrieval in the original model (a) and compressed models (b) and (c). The same query is paired with different documents, including a positive and a negative example.



(a) Baseline (b) Attention Drop (c) MLP Drop

Figure 7: Layer-wise query–document similarity for mean-pooling-based retrieval in the original model (a) and compressed models (b) and (c). The same query is paired with different documents, including a positive and a negative example.