Efficient Retrieval Augmented Generation with Channel Pruning and Dimensionality Reductions

Ty Perez
tyjperez@mit.edu

Finn Westenfelder
finnw@mit.edu

Yifan Yang
yifany@mit.edu

Abstract

Retrieval Augmented Generation (RAG) can significantly enhance Large Language Models (LLMs) by providing updated, verifiable, and domain-specific information. RAG is more cost-efficient than traditional fine-tuning and retraining methods but incurs a considerable initial computational cost to populate an embedding database as well as additional overhead for each retrieval. We explore two methods to accelerate the embedding and retrieval processes. First, we show channel pruning the bge-small-en-v1.5 embedding model reduces its size by 34% and latency by 16% while maintaining retrieval accuracy, substantially decreasing total embedding time. Second, we demonstrate the dimensionality of embeddings can be reduced up to 50% without significantly impacting retrieval accuracy. In addition to reducing the size of the embedding database, dimentionality reduction reduces the latency of cosine similarity computations.

1 Introduction

Retrieval Augmented Generation (RAG) is a popular technique for improving the responses of Large Language Models (LLMs) by supplementing the model's internal knowledge with external information. Previous work shows that RAG can provide an LLM with reliable, up-to-date information without the need for fine-tuning or retraining. RAG is particularly useful when users want to keep their data private, when data becomes outdated quickly, or when data is domain-specific. Further, RAG reduces inaccurate and irrelevant content generations, commonly referred to as "hallucinations," and improves LLM interpretability by providing users with the context of LLM generations [1, 2].

Standard RAG implementations use an embedding model for retrieval and a vector database for data storage [3]. Transformer-based embedding models, such as Contriever [4, 5], are standard for text retrieval. These models are trained on a large corpus of text and then fine-tuned on retrieval datasets, such as MS-MARCO [6]. When documents with semantic similarities are passed through the model, the resulting embeddings have high cosine similarity in vector space. RAG relies on the fundamental assumption that a query and the relevant information to answer that query have semantic similarity and thus have high similarity in the embedding space.

RAG is often separated into two phases: the embedding phase and the retrieval phase. In the embedding phase, every document of interest is partitioned into text chunks (based on a set token limit) converted into a sequence of word/token vectors and then processed by the embedding model to produce one vector embedding representing the entire text chunk. The resulting embeddings are stored in a vector database. In the retrieval phase, the user passes a query through the embedding model and the resulting query embedding is compared to the embeddings in the vector database using cosine similarity. The top k (user defined) embeddings based on cosine similarity are returned as the retrieval result. The documents corresponding to these embeddings are then appended to the query and passed to the LLM for generation. Finally, the LLM's response is returned to the user. While slight modifications to each stage of the RAG pipeline can improve retrieval in different contexts[7, 8],

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

we focus on a vanilla RAG implementation for our research. Figure 1 outlines the steps in a standard retrieval process.

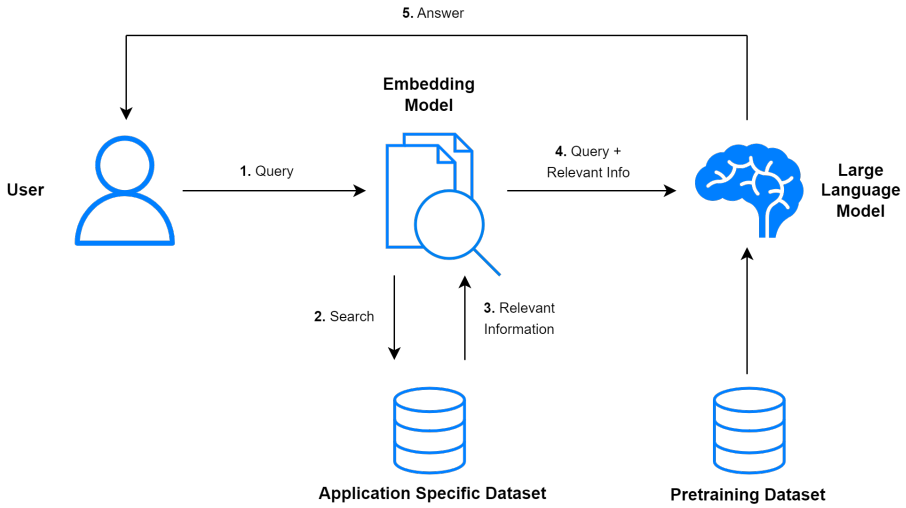


Figure 1: The steps in a standard RAG pipeline.

2 Motivation

The embedding and retrieval stages of RAG both benefit from efficiency improvements. The embedding stage incurs a large computational cost because every document of interest must be chunked and passed through the embedding model. For static datasets, initial embedding time increases linearly with the size of the dataset. For dynamic datasets, such as those comprised of social media feeds, news articles, financial data, or transportation data, the embedding stage must be repeated for each new data point in the dataset. Thus, small improvements in embedding model efficiency result in large improvements in overall embedding stage efficiency.

Moreover, the retrieval stage incurs a computational cost for embedding the query and performing document retrieval. Each query is passed through the embedding model and the resulting embedding is compared to every document embedding in the database to determine similarity. For normalized embeddings, computing cosine similarity is equivalent to taking the dot product of both vectors. Reducing the dimensionality of these vectors results in faster dot product computations. Since this computation is performed on every embedding in the database, small improvements in individual comparison efficiency result in large improvements in overall retrieval stage efficiency.

While these efficiency improvements are intuitive, the challenge resides in pruning the embedding model and reducing the dimensionality of the embeddings without sacrificing retrieval accuracy. Embedding models with fewer layers and smaller output dimensions can process data faster. However, they result in poor retrieval accuracy compared to larger models with more layers and larger output dimensions. Further, cosine similarity can be computed faster on smaller embeddings, but these embeddings retain less information about the original document. We aim to use channel pruning and dimensionality reductions on a large embedding model to increase the efficiency of both the embedding and retrieval stages while maintaining retrieval accuracy.

3 Methodology

3.1 Model, Dataset and Accuracy Metric

Embedding Model Our project used the bge-small-en-v1.5 embedding model, a small version of the bge-large-en-v1.5 model developed by the Beijing Academy of Artificial Intelligence [9]. The model achieves the highest accuracy on the Massive Text Embedding Benchmark (MTEB) [10] compared to all models with an embedding dimension less than 768. While we initially planned to

use the bge-large-en-v1.5 model, we were unable to fine-tune a model of this size due to hardware limitations. Instead we use the smaller bge-small-en-v1.5 model; a BERT [11] style model consisting of an embedding layer, 12 encoder layers, and a pooling layer, with 33.36 million parameters, a size of 133MB, and embedding dimensions of 384. We used a single V100 GPU for fine-tuning the model.

RAG Dataset We chose to use the MuSiQue (Multi-hop Questions via Single-hop Question Composition) dataset for fine-tuning and evaluating our model [12]. This dataset is well suited for RAG because it is comprised of a question, relevant information (positive paragraphs), irrelevant information (negative paragraphs), and an answer. For this work, we only use the initial queries of MuSiQue, and ignore the multi-hop compositions. In our RAG pipeline, the positive and negative paragraphs are first passed through the model and stored in a list of embeddings. This can be thought of as a very small embedding database. Then, the question is passed through the model and the resulting embedding is compared to the embeddings in the list using cosine similarity. The list is sorted by similarity score and the top k paragraphs are returned, where k is the number of positive paragraphs. Ideally, the returned paragraphs are the positive paragraphs. The returned paragraphs are appended to the query and passed to the LLM, which generates an answer to the question. Since our research focuses on retrieval accuracy, we do not evaluate the answer provided by the LLM.

Accuracy Metric To measure the accuracy of our model, we define a metric based on the position of the positive paragraphs in the sorted list. Ideally, the positive paragraphs have the highest similarity to the question, and are at the top of the sorted list. For example, if a data point has 16 negative paragraphs and 4 positive paragraphs, the sorted list has the positive paragraphs in the first 4 positions in the best case, and in the last 4 positions in the worst case. In this example, the best positions are index 0-3 and the worst positions are index 16-19. By calculating the sum of the best indices, worst indices, and actual positive paragraph indices we can define accuracy using equation 1.

$$accuracy = \frac{\sum worst\_positions - \sum actual\_positions}{\sum worst\_positions - \sum best\_positions} \tag{1}$$

3.2 Model Pruning and Fine-Tuning

Weight pruning [13] is a promising way to reduce the embedding model size and introduce sparsity in the model. If this sparsity can be exploited in hardware, it can lead to significant speedup in embedding generation process. We first plot the distribution of the weights in the embedding model, as shown in Figure 2. These weights are all from the Fully-Connected (FC) layers in the embedding model. As we can see from the plot, the weights are mostly concentrated around zero. This gives us the opportunity to prune the weights that are close to zero since they are not contributing much to the layer output.

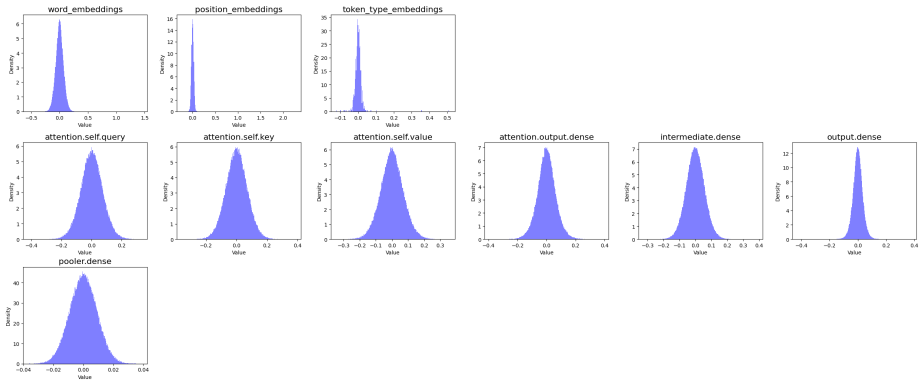


Figure 2: Weight distribution of FC layers of the Bert embedding model.

Fine-grain Pruning We first try to prune the weights in a fine-grain manner to see the pruning potential of the Bert model. Because fine-grain pruning poses the least restriction on the model

weight structure and are therefore less susceptible to accuracy loss. We use magnitude-based pruning, i.e. pruning the weights with the smallest magnitude, on all of the FC layer weights. Applying fine-grain pruning only, the model is able to maintain the accuracy w.r.t the dense model at 40% uniform pruning ratio per layer (shown in Figure 3).

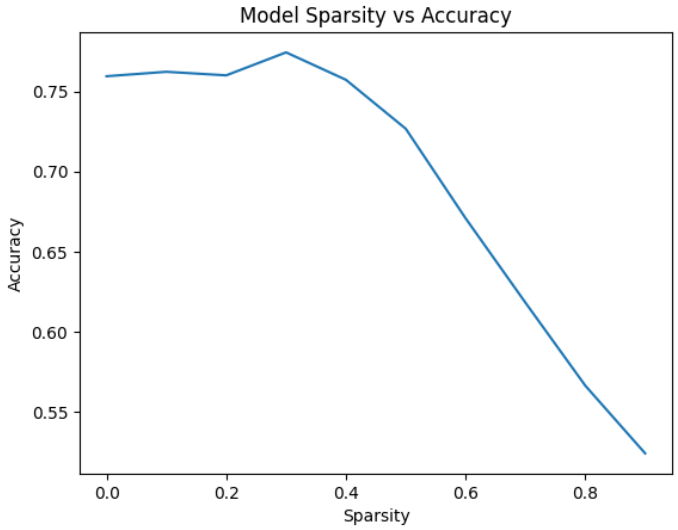


Figure 3: Embedding model accuracy at different pruning ratio w/o fine-tuning.

Fine-tuning Since fine-tuning can recover the model accuracy after pruning, we further add a fine-tuning step after pruning. Following the pruning lab practice, a per layer weight mask is generated based on the magnitude of the original weights. After every fine-tuning step (a mini-batch), the same mask is applied to the weights to prune the model. We fine-tune the model for 2 epochs to recover the accuracy loss from pruning. Thanks to the fine-tuning step, the model achieves 60%-70% pruning ratio while maintaining the accuracy.

Channel Pruning Though fine-grain pruning is able to achieve high pruning ratio (i.e. model size reduction), being able to leverage this sparsity in hardware to speedup the embedding generation process is another story. Existing hardware like GPUs are not able to take advantage of this kind of fine-grain sparsity in the model. Exploiting fine-grain sparsity requires hardware support [14]. Channel pruning of FC layers, on the other hand, can be effectively exploited in existing hardware, since the hardware effectively executes dense matrix multiplication with a smaller input dimension. Furthermore, the fine-grain pruning results indicate that there are indeed plenty of redundancy in the embedding model and there is an opportunity for pruning.

To this end, we apply channel pruning to the FC layers of the embedding model. More specifically, we prune the *input channels* of the FC layer weight. We rank each input channel weight by its Frobenius norm and prune the channels with the smallest norm.

Unlike fine-grain pruning study, pruning ratio for channel pruning is not uniform across layers. Because channel pruning is more susceptible to accuracy loss, we need to be more careful on pruning ratio to maximally preserve accuracy. Figure 4 shows the number of parameters in each FC layer. The two Feed-Forward Network (FFN) layers have the most parameters and their pruning ratio affects the model size reduction more. Therefore should receive more attention in channel pruning than other FC layers like projection. This insight is further strengthened by the sensitivity analysis in Figure 5. The figure shows that the FFN layers (blue line) are less sensitive to channel pruning than the other other non-FFN layers (orange line). This suggests that we can prune the FFN layers more aggressively than the non-FFN layers.

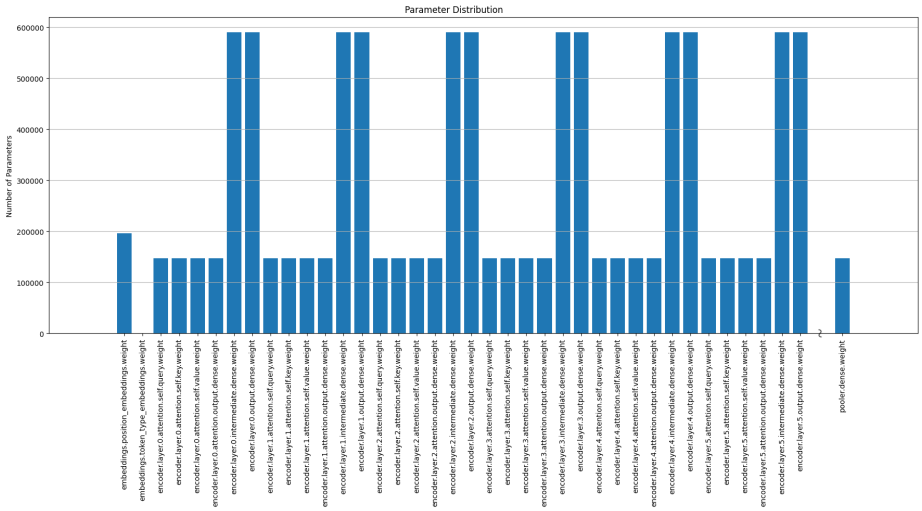


Figure 4: Number of parameters in each FC layer.

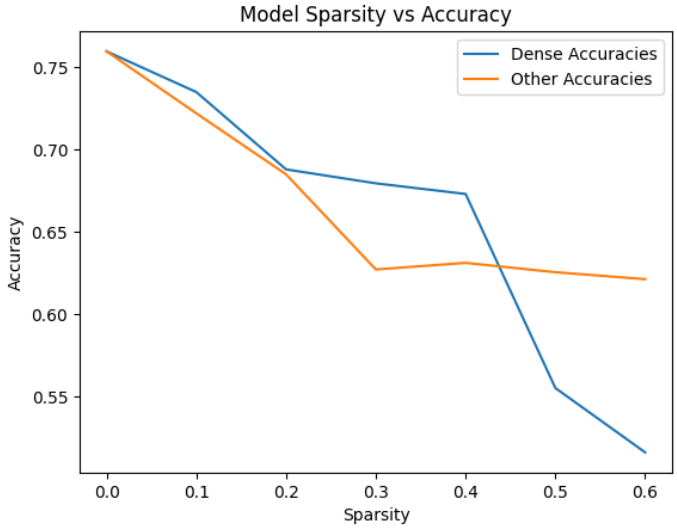


Figure 5: Embedding model accuracy at different FFN layer (Blue line: dense accuracies) and non-FFN layer (Orange line: other accuracies) pruning ratio w/o fine-tuning.

3.3 Leveraging Channel Pruning to Achieve Speedup

To achieve speedup of our in-channel-pruned model, we build a custom PyTorch Linear layer that transparently replace the vanilla Linear layer in the bert model without changing the architecture of the model and rearranging the parameters of other layers in the model. The custom Linear layer stores the pruned weight and input channel indices that survive channel pruning in the original weight. Then during the forward procedure, the custom linear layer only fetches the input activation tensor at input channel indices that survive channel pruning and multiply it with the pruned weight. In this way, the number of MAC reduction is proportional to the input channel pruning ratio. Moreover, the output of the custom Linear layer is the same as the vanilla Linear layer, so the rest of the model can be executed as usual. The innovation here is we condense both the input activation and weight at the input channel dimension, which is the dimension that we prune.

This custom Linear layer can be further optimized by conducting cascade channel pruning similar to the cascade token pruning in SpAtten [15]. This means the output channel of the linear layer can also be pruned based on the pruning mask of the input channel in the next FC layer. Cascade

channel pruning could result in quadratic MACs reduction compared to linear MACs reduction in our implementation. This approach is rather trivial to implement in CNN where the computation graph is more or less a linear chain. For transformer models, the computation graph is a more complicated DAG and the next layer of the current FC layer is not necessarily an FC layer. Incorporating this optimization requires case by case modification of the model architecture, which is feasible for a single Bert model but not scalable to other models. Therefore, we believe the current approach achieves good generalizability to arbitrary model architectures while sacrificing the potential quadratic MACs reduction.

3.4 Addressing Retrieval

In the standard retrieval process our query is passed through the embedding model and then compared to a database of text embedding using a similarity or distance scoring function, most often cosine-similarity. The top k most similar results from the database are then returned and used for generation.

Although relatively fast and straightforward this process faces several issues. First, it's not clear what information each embedding best represents from the original text chunk and the granularity of the representation will depend on the training set used to convey semantic similarity in the embedding model. This makes retrieval for highly specific or detail oriented questions particularly challenging. Additionally, it's not clear if or when query embeddings will be "similar" to positive retrieval text. In order to explore possible solutions to these issues, we analyze the embedding latent space of the MuSiQue dataset in section 3.5.

Second, RAG requires the storage and continuous querying of large vector databases. Retrieval accuracy can be improved by using a larger embedding model with more expressive high dimensional embedding, but longer embeddings require more storage; one million embeddings of size 1024 require about 4Gb of storage if stored as 32 bit floats. However, most text embeddings have been optimized for the representation of massive training sets that far exceed the scale and diversity of most dataset we might want to query with RAG. It's likely that for most domain specific datasets, standard text embeddings possess many redundant dimensions that can be pruned away. Intuitively we believe that an information efficient embedding length should be correlated with both the embedding text chunk size, as well as the diversity of content between text chunks in a database. In other words, embeddings for text chunks smaller than a model's context window likely carry redundant dimensions. Similarly, groups of text chunks with very little context diversity should not require a very high dimension embedding to be semantically distinguishable. In section 3.6 we explore how dimensional reduction affects retrieval accuracy.

3.5 Latent Space investigation

In our initial proposal we aimed to improve embedding context granularity by replacing embeddings of larger text chunks (ie. one or more paragraphs; parent-embeddings) with a composite function composed from embeddings from smaller chunks of the parent text (i e.one or several sentences; child-embeddings). We reasoned that 1) child-embeddings would cluster near parent-embeddings, and 2) some child-embeddings must be closer to the query embedding than their parent-embeddings. The proposed composite function we had designed (and tested) represented child-embeddings as a coarse grained gaussian density cloud in n-dimensional space. However this did not improve retrieval accuracy and after closer inspection of the spatial relationships between embeddings we were surprised to find that neither assumption 1 or 2 held true for the MuSiQue dataset.

Figure 6 shows 2D projections from several examples of embeddings for a Query (red start) and its corresponds Answer (green plus), positive paragraphs (blue diamond), negative paragraphs (orange diamonds), sentences from positive paragraphs (green dots), and sentences negative positive paragraphs (red dots).

Here we can see that sentence embeddings are not necessarily well clustered near their corresponding paragraphs and negative paragraph embeddings are often very close to the query. Additionally, it appears that answer embeddings aren't necessarily closer to positive paragraph embeddings than queries. All of these trends can be observed on a statistical level in figure 7 which plots the distributions of cosine distances between Queries or Answers and other paragraph or sentence embeddings.

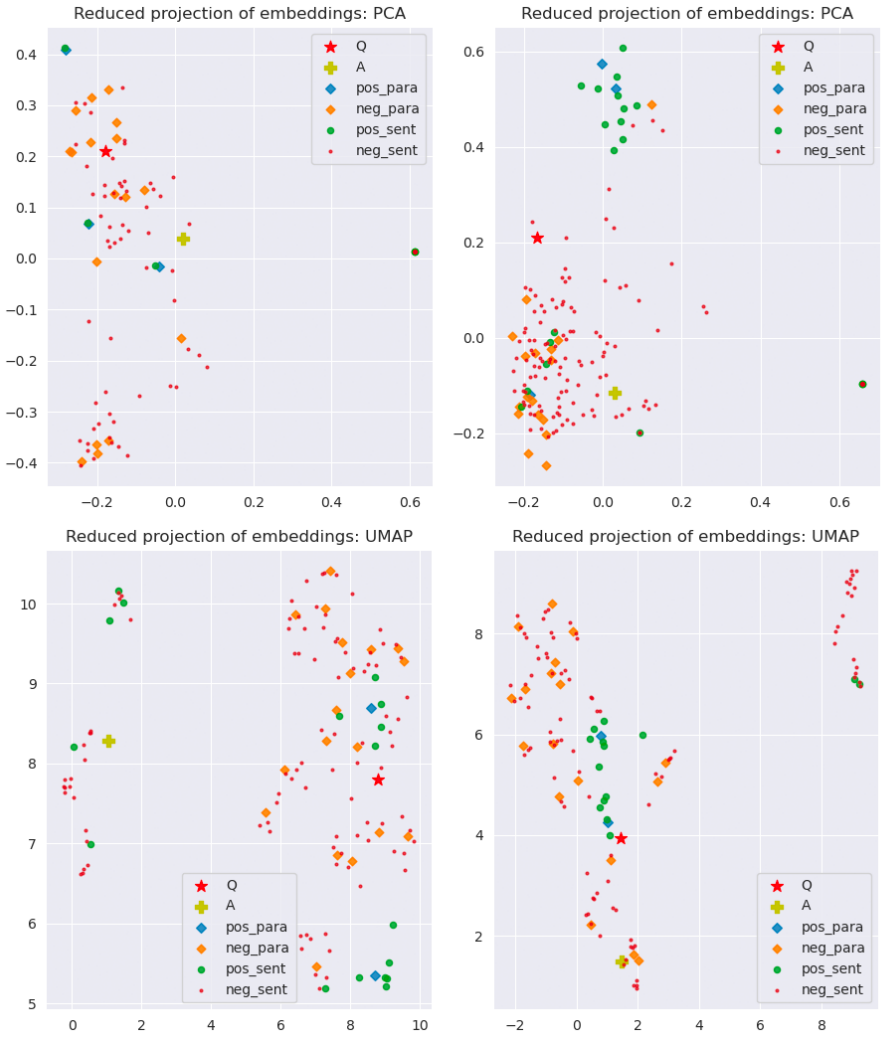


Figure 6: Visualization of embedding space for MuSiQue dataset several samples.

However, it should be noted that positive and negative text examples from the MuSiQue dataset are fairly short, usually between 4-10 sentences. Assumptions 1 and 2 may be more true for embeddings of text that completely fills the context window of an embedding model and where child-embeddings have more topical correlation.

Together these results suggest that a better way to improve retrieval accuracy may be to perturb the query representation rather than the embedding representation. One approach could be to use prompt engineering with a large language model to produce a new query text (or several) that resemble probable positive text, rather than an answer. A second possible solution may be to directly modify each query embedding vector using a Neural Network trained with a contrastive loss on positive/negative examples from the specific dataset being queried. We look forward to testing both of these approaches in the future.

3.6 Embedding Pruning

To address embedding size and redundancy several dimensionality reduction techniques were explored as methods of pruning: Linear PCA, Kernel PCA with a radial basis function, and a simple AutoEncoder (AE). Retrieval accuracy was tested for each compression method using the top 1,2,3 and 4 samples. Among these three approaches Linear PCA provided the best results by far.

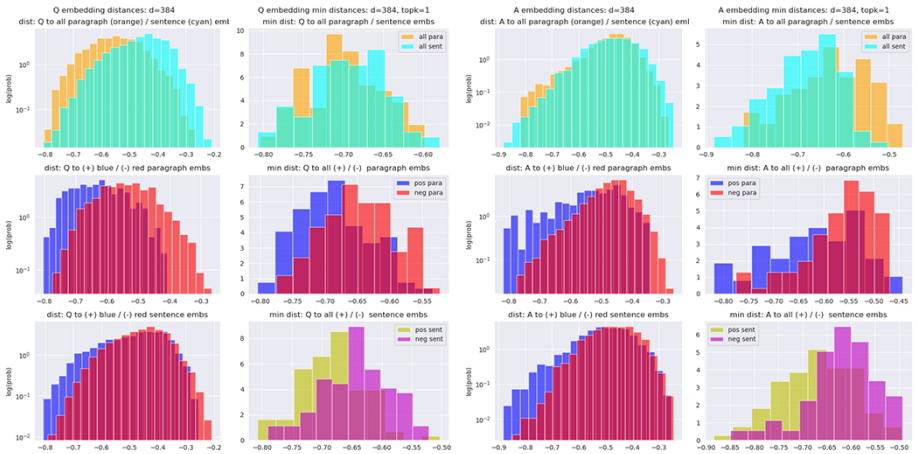


Figure 7: Distribution of cosine-distances from Queries or Answers and other embeddings.

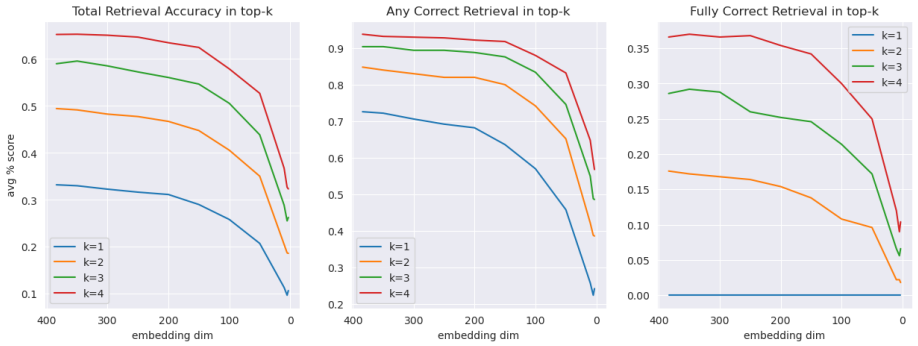


Figure 8: Retrieval accuracy as a function of PCA pruned embedding dimension.

Figure 8 shows that PCA dimensional reduction by up to 50% has minimal effect on retrieval accuracy. This was not the case for kPCA or AE which both resulted in retrieval accuracy of below 20% for a 25% pruning ratio. This is likely because these non-linear transformations can significantly alter the geometric relationships between embeddings, while linear operations tend to preserve existing cosine distance relationships.

Figures 91011 illustrate the effect of PCA at different dimensions on statistical distance relationships between embeddings. Here we can see that as we reduce the embedding dimension below 200 the distributions of cosine distances between query-positive text and query-negative text become less and less distingusihable.

The obvious next steps for this work is to apply well known quantization techniques to further reduce embedding size. Reducing our 32 bit floating point embeddings to 8 bits seems like a reasonable goal and would place our final compressed embeddings at 13% of their original size. In addition to reducing the storage size, both compression methods described here should reduce latency as well. In future work we hope to quantify this.

fp bit size	**Embedding dimension**	
	384	**200**
32	100.00%	52.08%
16	50.00%	26.04%
8	25.00%	**13.02**%
4	12.50%	6.51 %

Table 1: Embedding compression as a percentage of initial storage size.

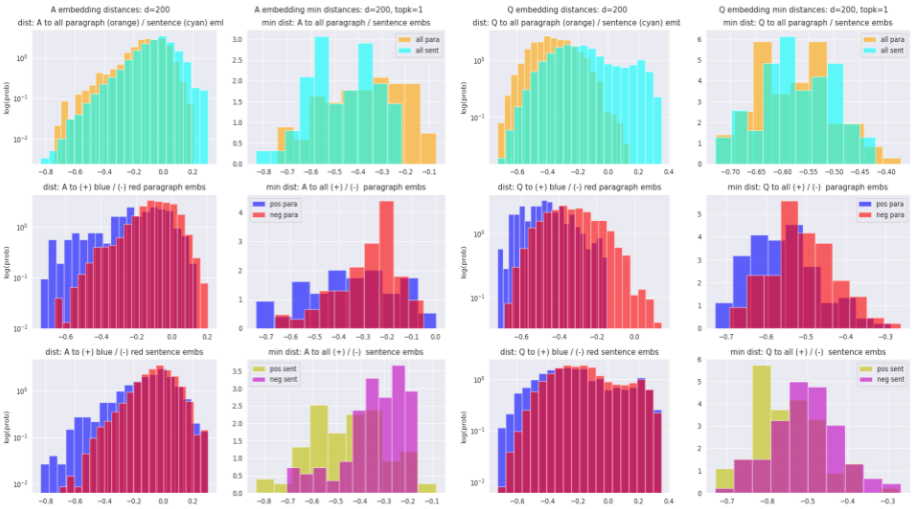


Figure 9: Distribution of cosine-distances after pruning to an embedding dimension of 200.

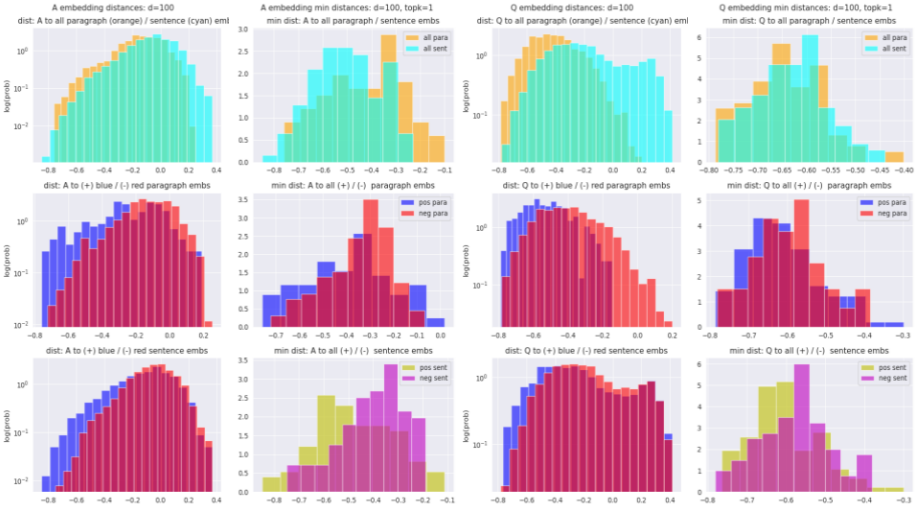


Figure 10: Distribution of cosine-distances after pruning to an embedding dimension of 100.

4 Results

Evaluation Methodology According to the channel pruning sensitivity study result, we've decided to prune 60% of the FFN layers and 30% of the non-FFN layers in the embedding model. Then we fine-tune the model for 2 epochs to recover the accuracy loss from channel pruning. The model is stored in FP32 format. The latency is measured using Nvidia T4 GPU available on Google Colab. We run the model inference on 500 validation samples from the MuSiQue dataset and report the overall latency.

Table 2 shows the comparison of the base model and the pruned model. Channel pruning is able to achieve a 34% model size reduction and 16% speedup on the embedding generation phase. After fine-tuning, the pruned model is able to recover most of the accuracy loss and achieves 69% accuracy compared to the baseline 76%.

Latency Breakdown To understand why the speedup of the pruned model is smaller than the model size reduction, we break down the latency of the embedding generation process through profiling. Figure 12a shows the latency breakdown of the base model. As we can see, the FC layers (ffn1, ffn2, attention_output, projection), which can benefit from channel pruning, only take up 40% of the total

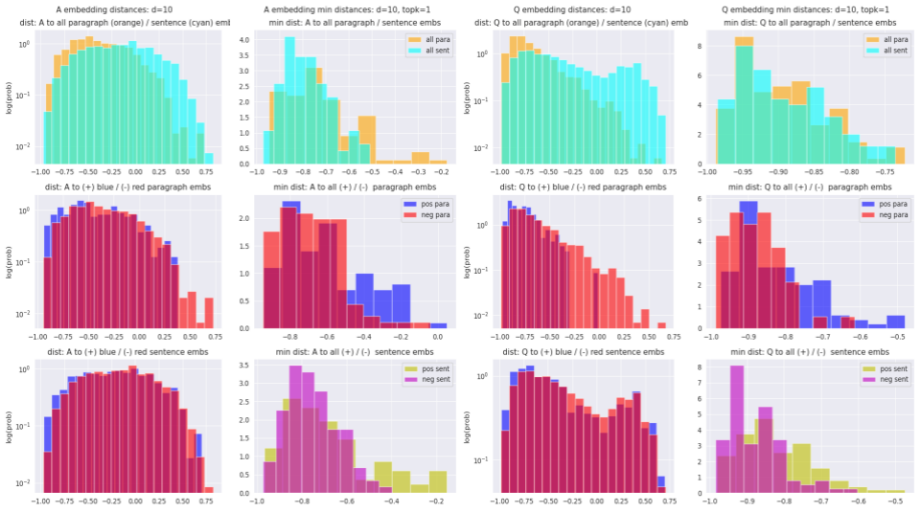


Figure 11: Distribution of cosine-distances after pruning to an embedding dimension of 100.

	Base Model	**Pruned Model**
Parameters	33.36 Million	22.18 Million
Size	133MB	88.8MB
Latency	61.0s	51.5s
Accuracy	76%	69%

Table 2: Comparison of Base and Pruned Embedding Model.

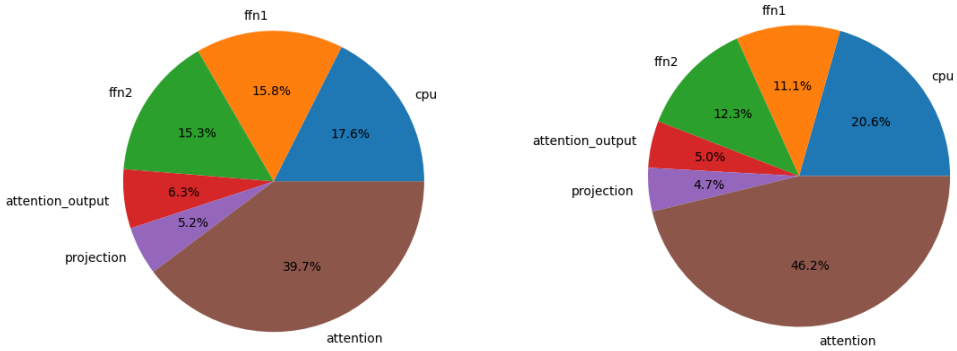
latency. The rest of the latency is spent on attention calculation and CUDA memory copy from CPU. The latency of these operations are not affected by channel pruning. Though the latency reduction of the FC layers should be the same as the model size reduction, the overall latency reduction is smaller due to the fact that FC layer latency only takes up a portion of the total latency. After pruning, as shown in Figure 12b, the FC layer latency only takes up 34% of the overall latency, proving the effectiveness of channel pruning on latency reduction.

Embedding Pruning In addition to reducing embedding model size, we found that linear PCA could reduce embedding dimensions by up to 50 percent without significant effect on retrieval accuracy. Although positive examples of retrieval were used to assess retrieval accuracy after dimensional reduction, they are not necessary to fit the PCA and prune embeddings. It's likely that careful analysis of embedding variance (or another statistical diversity metric) as a function of reduced embedding dimension could reveal what degree of pruning is optimal without requiring a set of positive/negative retrieval examples.

Retrieval Accuracy Although we were not able to make direct improvements to retrieval accuracy during this project, we were able to gain valuable insight as to why query based retrieval often struggles to return the best results. We believe that future work should focus on modifying the query to more closely resemble probably target text, but not necessarily probably answers. Prompt engineering and direct modification of the query embedding vector stand out as two promising avenues towards this goal that we intend to explore. Further, we expect modification of the query embedding to be fully compatible with PCA dimensionality reduction since the PCA is fit to target text embeddings alone.

5 Future Work

We hope to continue with this research and pursue publication. We have outlined the following tasks for future work:



(a) Base model. (b) Pruned model.

Figure 12: Latency breakdown on T4 GPU.

1. Measure the combined speedup from pruned model and dimensionality reduction.
2. Conduct a per layer sensitivity analysis of the embedding model. Currently we only compare the sensitivity of FFN layers and non-FFN layers.
3. Implement cascade channel pruning to achieve quadratic MACs reduction for FC layers.
4. Evaluate pruning attention layers in addition to FC layers.
5. Evaluate the FP16/BF16 model performance (currently using FP32) utilizing the Tensor Cores on the GPU.
6. Conduct channel pruning and dimensionality reduction on a larger embedding model, such as bge-large-en-v1.5, to determine if results are consistent.
7. Compare the retrieval accuracy of a large pruned model with a small un-pruned model.
8. Compare the retrieval accuracy of high dimension embeddings that have been reduced to low dimension embeddings that have not been reduced.
9. Evaluate model performance on additional datasets, such as the MS-MARCO, dataset to see if retrieval performance is consistent.
10. Implement known quantization methods on embedding vectors to further reduce size.
11. Explore statistical analysis of embedding pruning as a means of finding optimal pruning ratio for a given dataset.
12. Prompt engineer query text to output multiple modified query text examples that resemble probably target text.
13. Test direct modification of query embedding vectors by simple addition of small neural network modulation.

References

[1] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. *CoRR*, abs/2005.11401, 2020.

[2] Huayang Li, Yixuan Su, Deng Cai, Yan Wang, and Lemao Liu. A survey on retrieval-augmented text generation. *CoRR*, abs/2202.01110, 2022.

[3] Yutao Zhu, Huaying Yuan, Shuting Wang, Jiongnan Liu, Wenhan Liu, Chenlong Deng, Zhicheng Dou, and Ji-Rong Wen. Large language models for information retrieval: A survey, 2023.

[4] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Towards unsupervised dense information retrieval with contrastive learning. *CoRR*, abs/2112.09118, 2021.

[5] Peitian Zhang, Shitao Xiao, Zheng Liu, Zhicheng Dou, and Jian-Yun Nie. Retrieve anything to augment large language models, 2023.

[6] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. MS MARCO: A human generated machine reading comprehension dataset. *CoRR*, abs/1611.09268, 2016.

[7] Langchain framework. `https://github.com/langchain-ai/langchain`, 2023.

[8] Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels, 2022.

[9] Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. C-pack: Packaged resources to advance general chinese embedding, 2023.

[10] Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*, 2022.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[12] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multi-hop questions via single-hop question composition. *CoRR*, abs/2108.00573, 2021.

[13] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[14] Eric Qin, Ananda Samajdar, Hyoukjun Kwon, Vineet Nadella, Sudarshan Srinivasan, Dipankar Das, Bharat Kaul, and Tushar Krishna. Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 58–70. IEEE, 2020.

[15] Hanrui Wang, Zhekai Zhang, and Song Han. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 97–110. IEEE, 2021.