EMBEDDINGALIGN RAG: BOOSTING QA SYSTEMS *

Pierre-Louis Biojout
phospho
plb@phospho.ai

Wandrille Flamant
phospho
wandrille.flamant@phospho.ai

Frederic Legrand
phospho
frederic@phospho.ai

Nicolas Oulianov
phospho
nicolas@phospho.ai

Paul-Louis Venard
phospho
paul-louis@phospho.ai

ABSTRACT

The Retrieval-Augmented Generation (RAG) method improves question-answering by retrieving relevant information before generating answers. In this paper, we introduce EmbeddingAlign RAG, a new way to improve document retrieval in RAG systems. In this work we demonstrate that applying a linear transformation to both the query embedding and the chunks embeddings produces a significant improvement in retrieval accuracy. Our approach uses small datasets, such as a single PDF with 500 chunks, an out-of-the-box model embedder (can be a black-box embedding model such as the OpenAI embedding API), and leverages already existing embeddings collections. The model is trained on CPU with standard hardware and increases retrieval inference time by less than 10ms. We show that EmbeddingAlign RAG improves retrieval accuracy, boosting the hit rate from 0.89 to 0.95. The simplicity of implementing this method makes it ideal for direct applications in real-world Q&A systems.

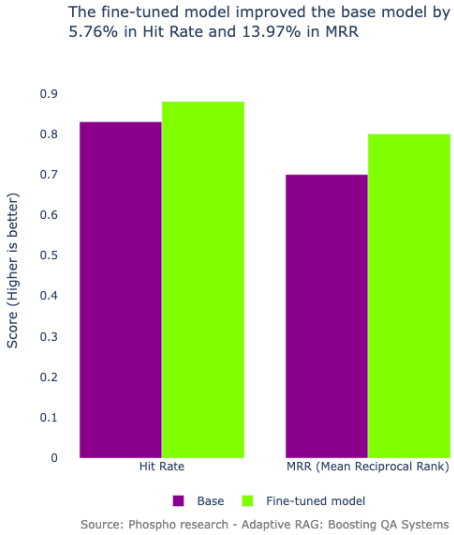

Figure 1: Performance comparison (base model vs EmbeddingAlign fine-tuning)

Keywords Text Embedding · Retrieval Augmented Generation (RAG)

* *Citation*: **Pierre-Louis Biojout, Wandrille Flamant et al. EmbeddingAlign RAG: Boosting QA Systems. Technical Report, phospho, 2024.**

1 Introduction to RAG and Embedding Optimization

1.1 Challenges of existing RAG systems

A RAG system combines a retriever and a generator (like GPT-4). In this system, the retriever fetches relevant documents or document chunks based on user queries (questions). This system usually relies on document embeddings and query embeddings to retrieve relevant chunks. Periodically, new documents are added to the knowledge base to address expanding use cases.

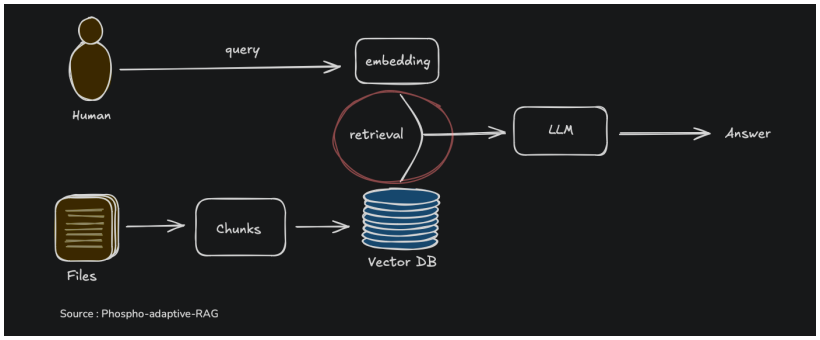

Figure 2: Classic RAG pipeline

The mathematical representation of the RAG system is as follows: Let Q be the set of user queries, D be the set of document chunks, and E be the embedding function. For a query $q \in Q$ and a document chunk $d \in D$, the retrieval score $S(q, d)$ is typically calculated as: $S(q, d) = \text{similarity}(E(q), E(d))$ where similarity is often cosine similarity: $\text{similarity}(a, b) = \frac{a \cdot b}{|a||b|}$

However, as more documents are introduced, the retriever may pull irrelevant chunks, leading to poor performance in retrieval. A reason for this decrease in performance is because existing embeddings (either for the queries or chunks) are not well-aligned with the new data.

Furthermore, trying or training a new embedding model is expensive, as large document embeddings collection are costly and difficult to update from an engineering perspective. This leaves many RAG systems in a dead-end, where retrieval needs to be improved to push performances, but it's too costly to do.

1.2 Our Approach: Embedding Alignment via Linear Transformation

We propose training a single linear transformation that will be applied on embedding vectors of both query and text chunks embedding vectors. The goal is to learn a transformation that brings the embeddings of user queries closer to the most relevant document chunks, improving retrieval accuracy.

Let T be the linear transformation matrix we want to learn. The new retrieval score becomes: $S'(q, d) = \text{similarity}(T \cdot E(q), T \cdot E(d))$, where:

- $T \in \mathbb{R}^{N \times N}$
- $E(q), E(d) \in \mathbb{R}^N$

In our case, $N = 1536$ (the dimension of the embeddings from *text-embedding-3-small* by OpenAI).

To find a matrix T that improve the retrieval quality of our RAG pipeline, we need the following data:

- Query embeddings (user questions)
- Good chunk embeddings (relevant chunks from documents)
- Distractor chunk embeddings (irrelevant or misleading chunks)

In an already deployed RAG system, these data can be easily collected through feedback mechanisms or annotated datasets. For example, users or annotators' feedback on end results (thumbs up/down on results) can be used to define good chunks and bad chunks.

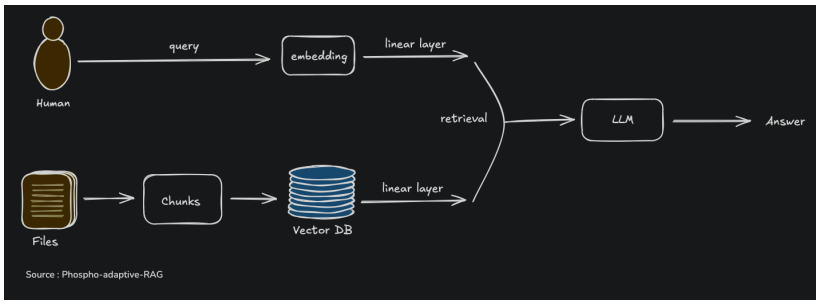

Figure 3: Our RAG pipeline

From an inference perspective, the linear layer needs to be applied to the user query embedding during the Retrieval step, but also to all of the embeddings stored in the vectorstore. For latency and compute efficiency considerations, we recommend applying the linear transformation to all of the vectors of the vectorstore once, before starting to perform retrievals.

1.3 Synthetic Dataset Generation

In our case, we don't have access to proprietary production data. To overcome this limitation, we simulate a realistic scenario by generating a synthetic dataset from existing documents (representing the knowledge base). This approach is also useful in cases where production data is absent or insufficient, as one only needs the text chunks of the knowledge base to improve the RAG.

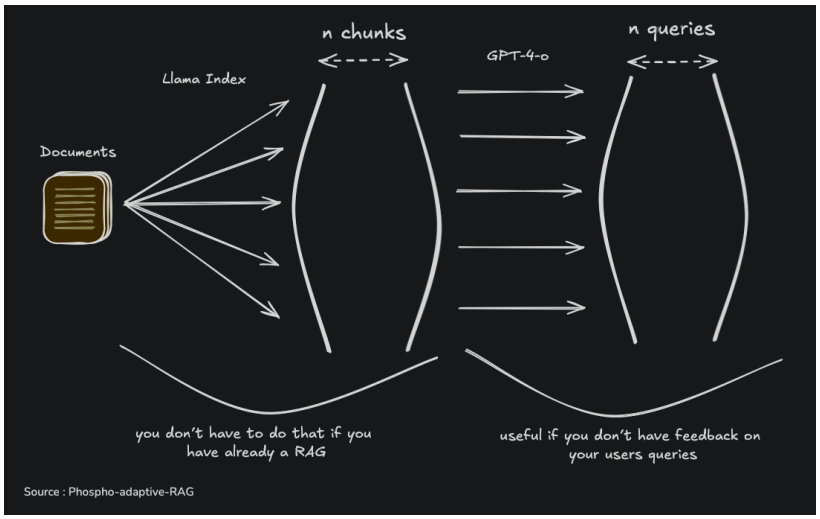

Figure 4: QA pairs generation

We base our synthetic dataset on publicly available documents and models:

- Documents: SEC Form 10K reports from Uber and Lyft, which contain rich text on similar business activities (mobility services).
- Models: GPT-4o is used to generate queries and LlamaIndex to generate document chunks. For each chunk, a query (question) is generated to form (query, chunk) pairs.

Below is an example of a (query, chunk) pair from our dataset:

Query (generated question)	Chunk (original document)
How is the commercial agreement for the utilization of Lyft rideshare and fleet data by Woven Planet accounted for, and what financial impact did it have on the company's deferred revenue?	the consolidated statement of operations for the quarter ended September 30, 2021. The commercial agreements for the utilization of Lyft rideshare and fleet data by Woven Planet is accounted for under ASC 606 and the Company recorded a deferred revenue liability of $42.5 million related to the performance obligations under these commercial agreements as part of the transaction at closing. The Company also derecognized $3.4 million in assets held for sale.97

Then, both the queries and chunks are embedded using the OpenAI *text-embedding-3-small* model. In the case of an already deployed RAG system, the embeddings of the chunks would already exist and be stored in a vectorstore.

With the Uber document, we generate 686 embedding pairs for training and validation dataset. With the Lyft document, we generate 818 embedding pairs, used as the test dataset. The cost of generating this dataset is modest.

2 Data preparation & augmentation

To train the Linear transformation model that aligns user queries with the corresponding text chunks, we use triplet loss. This loss brings the correct pairs (query and chunk) closer together and pushes the incorrect chunks further apart. In addition, generating triplets significantly increases the size of the training dataset virtually for free.

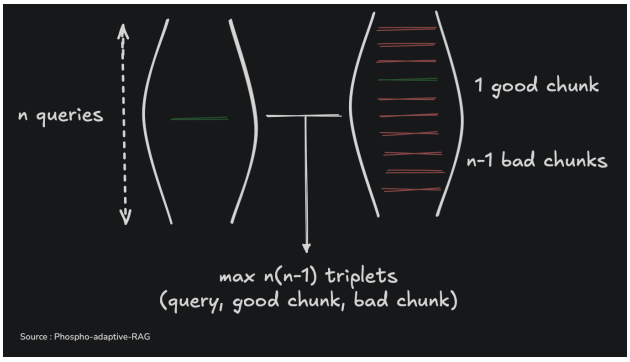

Figure 5: Illustration of the augmentation process

A triplet consists of three elements:

- Query (question embedding)
- Chunk (correct document chunk embedding)
- Distractor (an incorrect document chunk embedding)

Each pair of (query, chunk) is matched with several distractors, which are incorrect chunks. We call **augmentation factor** the ratio of incorrect chunks associated with each pair.

With an augmentation factor of 0.3, we create 141 100 triplets for the train-val dataset.

By controlling triplets, we can penalize specific associations that one may want to avoid, further refining retrieval performance. Such distractors could be for instance collected through a user feedback mechanism (thumbs up or down buttons).

3 Training process

To improve the retrieval performance, we train a linear layer over the embeddings to minimize the triplet loss. The **triplet loss** encourages the model to minimize the distance between the query and the correct chunk while maximizing the distance between the query and the distractor.

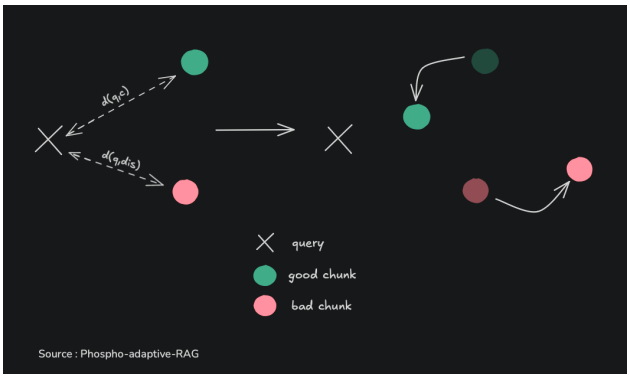

Figure 6: Illustration of the triplet loss effect in the embedding space

The triplet loss formula is:

$$\text{Loss}(q, c, dis) = \max(d(q, c) - d(q, dis) + \text{margin}, 0)$$

Where:

- q is the query embedding,
- c is the correct chunk embedding,
- dis is the distractor chunk embedding,
- $d(x, y)$ represents the distance between two embeddings. In our case, we use the normalized cosine distance.
- $margin$ is an hyperparameter. It's a positive value that ensures the model creates sufficient separation between correct and incorrect pairs.

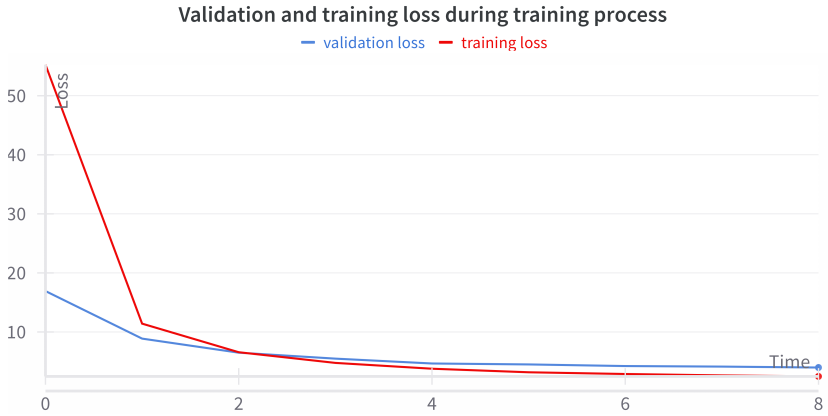

Figure 7: Training and validation losses over time

For training, we only rely on **consumer-grade CPU** and **do not use GPU**, as the linear layer and cosine distance require little computing resources.

4 Results

We compare our adapted user query embedding (that use the trained Linear transformation) to a simple one for the retrieval of document embeddings using two metrics:

MRR (Mean Reciprocal Rank): Measures how highly the correct answer is ranked, with higher MRR indicating better ranking performance.

$$MRR = \frac{1}{card(Q)} \sum_{q \in Q} \frac{1}{rank_{\text{retrieve(q)}}}$$

where $rank_{\text{retrieve(q)}}$ is the position of the first relevant item for query q in the top-k results.

Hit Rate: The percentage of queries where the correct answer appears in the top-k results, reflecting overall retrieval quality.

$$HR = \frac{1}{card(Q)} \sum_{q \in Q} \mathbb{1}_c(retrieve(q))$$

where $\mathbb{1}_c(retrieve(q))$ is the indicator function defined as:

$$\mathbb{1}_c(retrieve(q)) = \begin{cases} 1 & \text{if the chunk paired with the query } q \text{ appears in the top-k results} \\ 0 & \text{otherwise} \end{cases}$$

In other words, the hit rate calculates the proportion of queries for which at least one correct answer is retrieved within the top-k results. This metric provides a measure of the retrieval system's ability to surface relevant information for a given query within a specified number of top results.

The value of k is typically chosen based on the specific requirements of the RAG system, such as the number of results shown to the user or the number of chunks processed by the generation step. Common values for k include 1, 5, or 10, depending on the application.

Please note that the absolute value of these scores depends on the chosen value for k.

We mostly use the Hit Rate for the top 4 results ($k = 4$) and MRR as they best highlight improvements in retrieval accuracy.

Metric	Simple embedding (reference)	Adapted embedding (ours)
Hit Rate	0.89	0.95
MRR	0.69	0.83

Table 1: Comparison of Simple and Adapted Embeddings

The adapted embedding shows a significant improvement on both metrics.

5 Impact on inference time

On our standard hardware, the model increases the retrieval time by 8.6% (less than 10ms). In practice, this delay is barely noticeable, as the later generation step in RAG may take several seconds.

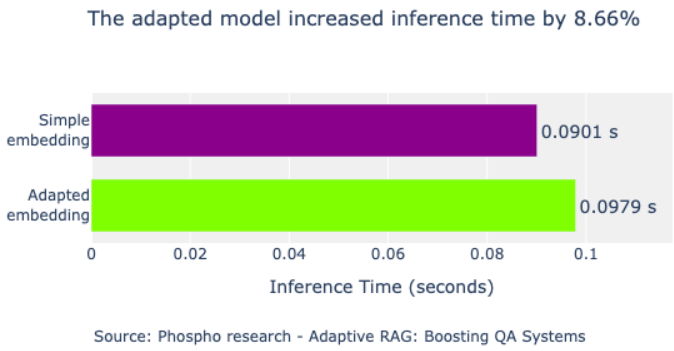

Figure 8: Impact on inference time

6 Conclusion

In this paper, we introduce EmbeddingAlign RAG, a new way to improve document retrieval in Retrieval-Augmented Generation (RAG) systems. By adapting the embeddings to include domain-specific context, we significantly improve retrieval accuracy at a low computational cost by leveraging existing embeddings. Our method is efficient, scalable, and easy to implement. This makes EmbeddingAlign RAG ideal for practical question-answering systems, especially with smaller datasets, as it improves performance while increasing retrieval by less than 10 ms.