Embedding model selection

Introduction

Background

The model used initially to generate the embeddings was Qwen 2.5 – 1.5B. At first, it was unclear whether this model was capable of representing text or chunked information using vectors.

The aim was to analyze whether this model was capable of capturing this type of information, then to find a way to compare it with other embedding models and determine which was the best, based on our performance in generating the vectors....

After the initial chunking of the first set of documents, a retrieval test was carried out, during which it became apparent that the documents cited as relevant in the response did not have a clear interrelationship, raising the first doubts about the quality of the embeddings.

How are embeddings generated in the model?

There are several techniques for generating embeddings; in this case, after some research, we found that this model uses the 'Last Token' method, whereby the final token is assumed to contain all the preceding information, and the generated embedding reflects this.

This is not the right way to tackle this problem because the context of the rest of the chunk is lost, and when performing a retrieve, the information from the last token is not sufficient; this information can be found at the following link.

Alternatives

In order to select the best model for generating embeddings, we conducted a search to find a model that specialised in embeddings, was lightweight—to meet the VRAM limitation (4 GB)—and was well-regarded by the market.

To this end, we chose to use the well-known MTEB (Massive Text Embedding Benchmark) leaderboard, which compares over 100 text and image embedding models across more than 1,000 languages and domains, and across multiple tasks such as retrieval, re-ranking and classification. Given our domain, we chose to search for models and rank them based on the scores obtained in the retrieval task, focusing on the programming domain.

After analysing the leaderboard, we decided that the best option would be Qwen3-Embedding-0.6B. It is not the highest-ranking embedding model, but given its characteristics and our circumstances, we felt it was the best choice. Not only is it a model from the Qwen family (which may be beneficial for our generator model, which is also Qwen), but it is also a small model (1GB VRAM and 0.596 billion parameters) and is one of the few to have such a large context window (32,768) whilst being so compact. All this while achieving a retrieval score of 90%.

Our approach

Evaluation datasets

Since the first model provided to us (Qwen2.5 1.5b) does not appear on this leaderboard, we decided to evaluate both models and obtain metrics to support our decision. To do this, we evaluated both models using three datasets:

- CodeXGlue: A task-oriented benchmark for code understanding and generation, containing real code from repositories (primarily GitHub).
- Scifact: SciFact, a dataset containing 1,400 scientific statements written by experts. Although this dataset does not focus on code, we thought it would be a good idea to try it out in another domain.
- CoSQA: One of the most challenging datasets, CoSQA (Code Search and Question Answering) is a large-scale dataset containing over 20,000 real, human-annotated pairs of natural-language queries and Python code snippets, compiled from Bing search logs.

The election of these datasets could be non-optimal for the problem at hand – AVAP documentation and examples - but we think that one of these datasets, CoSQA, is probably not too far semantically so that its results could be valid to our problem.

Metrics

In order to evaluate the models, we need a set of metrics that provide a quantitative measure of how 'good' or 'bad' the generated embedding is, so that we can make a data-driven decision.

These evaluations are carried out by feeding the model a series of queries, predefined in the datasets mentioned earlier, and then comparing the model's response against the ground truth.

Currently, the most widely used metrics on the market are as follows:

- NDCG@k: Normalised Discounted Cumulative Gain; this takes into account both relevance and the position in which the document appears. It is the most widely used in rankings, and the most representative. In this case, this provides a value between 0 and 1 for the importance of the documents in the response, taking into account their order and adding their relevance; this will penalise heavily if a relevant document is not the first, and only slightly if an irrelevant one is the tenth.
- MAP@k: stands for Mean Average Precision; in this case, what matters is how many of the relevant documents are retrieved and their position within the top k. This places a high value on documents that are relevant and appear higher up in the top k.
- Recall@k: of all the relevant documents, this tells us how many of them are included in the top k.
- Precision@k: of the k results returned by the model, this tells us how many of them are correct.

These metrics are ultimately normalised between 0 and 1, so those with a higher value do not necessarily indicate that the evaluation of the embeddings is better, which would make that model more suitable.

Benchmark libraries

To obtain results, we used the following libraries to run the benchmarks:

- MTEB: MTEB is not just a leaderboard; it also includes a library for running benchmarks to evaluate the generated embeddings. Due to the long execution times, it was discarded and replaced by BEIR.

- BEIR: BEIR (Benchmarking IR) is a benchmark focused exclusively on information retrieval. Given a query, it aims to retrieve the relevant documents. Unlike MTEB, BEIR is more focused on retrieval tasks, which is why we believe it would be a good alternative benchmark. It does not evaluate embeddings directly, but it does so indirectly.

Evaluation results

CodeXGlue

- Qwen2.5-1.5B

k	NDCG	MAP	Recall	Precision
1	0.00031	0.00031	0.00031	0.00031
3	0.00061	0.00051	0.00088	0.00029
5	0.00086	0.00065	0.00151	0.00030
10	0.00118	0.00078	0.00250	0.00025

- Qwen3-Embeddings-0.6B

k	NDCG	MAP	Recall	Precision
1	0.9497	0.9497	0.9497	0.94971
3	0.9695	0.9650	0.9825	0.32750
5	0.9716	0.9662	0.9876	0.19752
10	0.9734	0.9669	0.9929	0.09930

Scifact

- Qwen2.5-1.5B

k	NDCG	MAP	Recall	Precision
1	0.02333	0.02083	0.02083	0.02333
3	0.03498	0.03083	0.04417	0.01556
5	0.04040	0.03375	0.05750	0.01267
10	0.04619	0.03632	0.07417	0.00833
100	0.07768	0.04123	0.23144	0.00277

- Qwen3-Embeddings-0.6B

k	NDCG	MAP	Recall	Precision
1	0.56333	0.52994	0.52994	0.56333
3	0.64367	0.61170	0.70350	0.25889
5	0.66577	0.62815	0.75967	0.17067
10	0.68551	0.63830	0.81611	0.09300
100	0.71285	0.64466	0.94000	0.01070

Cosqa

- Qwen2.5-1.5B

k	NDCG	MAP	Recall	Precision
1	0.00000	0.00000	0.00000	0.00000
3	0.00000	0.00000	0.00000	0.00000
5	0.00000	0.00000	0.00000	0.00000
10	0.00000	0.00000	0.00000	0.00000
100	0.00210	0.00043	0.01000	0.00010

- Qwen3-Embeddings-0.6B

k	NDCG	MAP	Recall	Precision
1	0.17400	0.17400	0.17400	0.17400
3	0.27374	0.24700	0.35200	0.11733
5	0.33509	0.28080	0.50200	0.10040

10	0.39086	0.30466	0.67000	0.06700
100	0.45099	0.31702	0.95200	0.00952

Discussion

Following a comparative analysis of metrics and embedding methodologies, the qwen-3-emb-0.6B model was selected to replace qwen-2.5-1.5B. This decision is supported by its consistently higher performance levels across all tested benchmarks.

All results and notebooks are under the research/embeddings folder

Bibliography:

https://github.com/beir-cellar/beir

https://huggingface.co/spaces/mteb/leaderboard

https://github.com/Jun-jie-Huang/CoCLR

https://huggingface.co/datasets/google/code_x_glue_tc_nl_code_search_adv

https://huggingface.co/datasets/allenai/scifact