Final Report For Inter IIT 11.0

Abstract :

After the mid evaluation we had to change our approach since the hardware constraints were not lifted. We went with a three step pipeline for retrieving the most similar passages , reranking those passages based on semantic similarity and then extracting answers from the reranked passages as per the query. All these steps use transformers so we parallelized each of the tasks on the 2 CPU cores available on the free tier colaboratory and achieved great results.

Implementation :

We tried a lot of approaches after the mid evaluation whose reports will be present in the ablation study. In essence our approach is a three step procedure as mentioned earlier.

1) Paragraph Retrieval :

- **Context/Query Encoder**: We tried various models and methods for producing encodings. A few notable methods are , word2vec averaging , fasttext , doc2vec , wiki2vec. These compute efficient methods were unable to capture the semantics of the underlying contexts/queries which was apparent from their performance in the retrieval task. Hence we shifted to transformer based models which even though being compute heavy , have architectures and pretraining procedures designed to capture semantics and understand natural language better. For this we tried Sentence BERT , Google Universal Sentence Embeddings and a ROBERTA based model for generating document and query embeddings. The Google Universal Sentence Embeddings performed the best overall and all the upcoming metrics are reported using this as the encoder.

- **Similarity Metric**: The similarity metric that we used was cosine similarity , we also tried using euclidean distance but it was slower and performed worse than the former method. Finally we store the encoder representations of the query and contexts in a vector database after which we compute the dot product of normalised query and context embeddings (a given query with each context from that theme) which amounts to the cosine of the angle between them. The higher the similarity the more is the probability that the query is answerable by that particular context / similar to that context. Hence we select the top k highest similarity contexts for each query and pass them to the re-ranker model.

S. No.	Method Name	Latency (in secs)	Accuracy
1.	Word2Vec Averaging	0.002	38%
2.	FastText	0.002	44%
3.	Doc2Vec	0.0025	52%
4.	**Google Universal Sentence Encoder**	**0.2**	**74%**
5.	Sentence BERT	0.35	60%

☐ Experiments With Retriever:

Along with the proposed retrieval pipeline we also performed several different experiments. We majorly focused on improving the embeddings generated by the encoder to better the given data. The two major approaches we took were:

- **Contrastive Learning:** This is a fairly common method used in NLP to cluster embeddings. We looked at the paper "Unsupervised Document Embedding via Contrastive Augmentation" by Luo et al. for a hint at the training procedure. We fixed context embeddings as anchor points and used the Contrastive Loss to bring the relevant question embeddings closer in the latent space.

- **Classification with Adaptive Margin Regularisation:** We were heavily inspired by the paper "Adaptive Margin Diversity Regularizer for handling Data Imbalance in Zero-Shot SBIR" by Soma et al. to deal with the data imbalance. The idea of the paper was to train models which share latent spaces , i.e in this case to have the query and context embeddings in the same latent space thus increasing the similarity between semantically similar query and context pairs. It does this by adding a regularisation term to the normal training objective which is based on the weight matrix that performs the classification in the final layer and the number of samples we have for each class from the two different datasets whose encoded representations must share a latent space. However promising the results of this paper were , we were unable to find the right hyperparameters that are required for it to work as intended in our case.

  We believe both these experiments have promising prospects for improving any retriever's accuracy and if provided with more data and time they would definitely give better results.

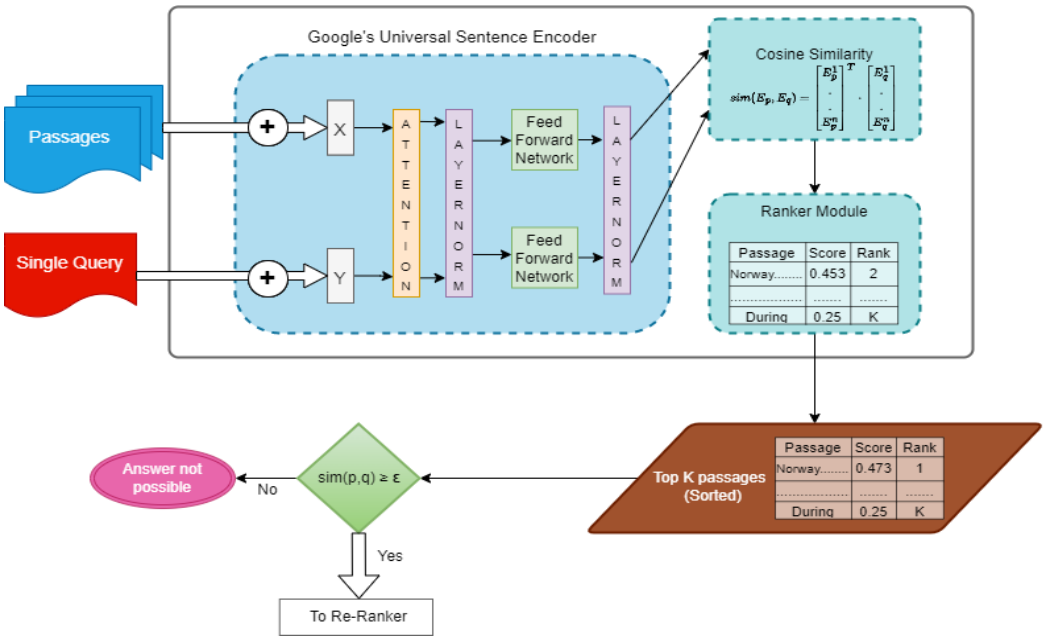

Fig 1. Retriever Model

2) Re-Ranking Module:

- The vanilla retriever model performed well and gave a final reportable metric of about 0.68. To improve this we looked at the retrieved paragraphs and found out that most of the ground truth paragraphs were not actually ranked very high in the retrieved list of contexts. To overcome this we decided to introduce a new cross-encoder model which looks at both the question and the retrieved contexts and re-ranks them according to whether or not they can be answered or not. This provided a significant boost to our performance metric. The advantage of the re-ranking model is that it is able to perform self-attention across both the context and query to rank the most relevant passage the highest.

- We tried 3 different cross encoders, namely, MiniLM, Roberta Tiny and Tiny BERT. Tiny BERT proved to be the fastest which is what we wanted considering the hardware constraints.

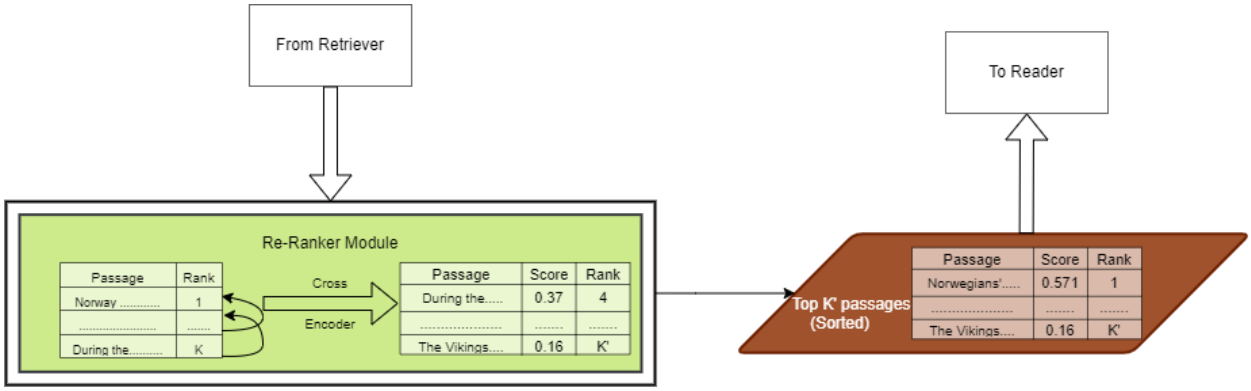

Fig 2. Re-Ranker Module

3) Reader Module:

- The final step of the pipeline was to set up a model which extracts the answer texts from the retrieved and re-ranked passages. This model was the same one we used during the mid-eval as we had already run quite a few experiments using it and it gave the best result i.e. it was fast and the one with the best metric.

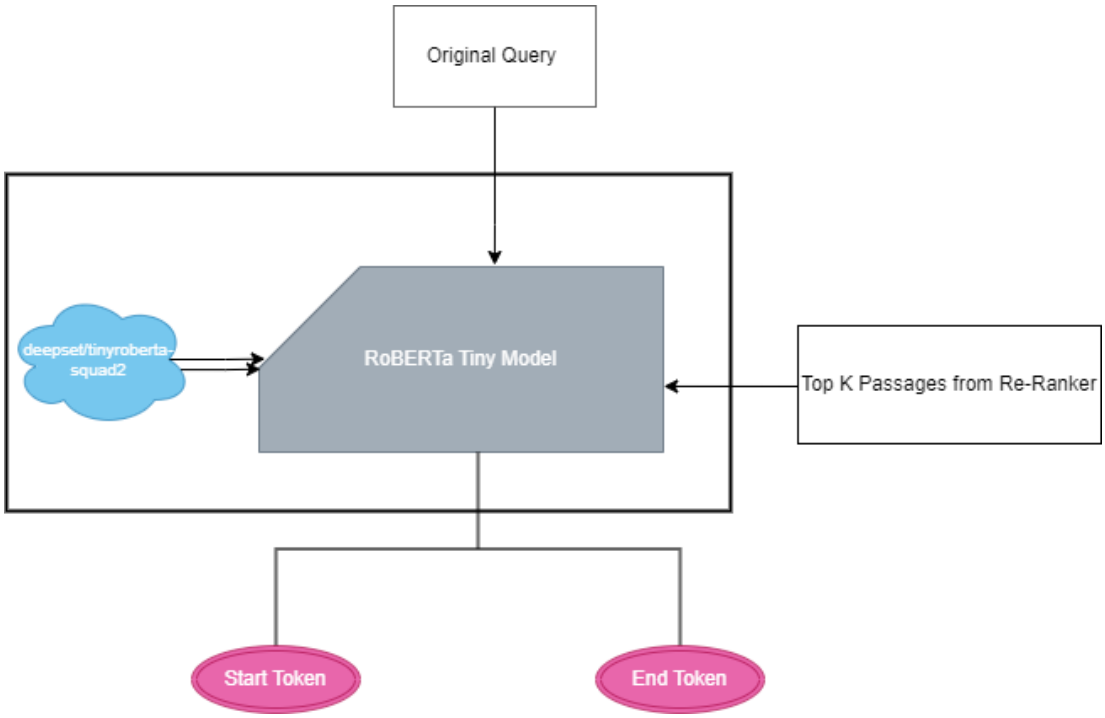

Fig 3. Reader Module

Optimisations:

As the metric heavily penalises latency in the modules and we couldn't use any hardware accelerators we found different ways to optimise inference times so that we could run the compute heavy transformer models efficiently on the CPU.

1) Choice of Language Models Used:

- We only tried using light weight models such as MiniLM and Tiny BERT. This was due to the fact that their inference times on CPU were almost 50x lesser than their larger counterparts and were only slightly behind in the reported metrics. This worked well for us as the latency-performance trade-off was good enough for the task at hand. We tried several light models such as:
    - MiniLM
    - Tiny BERT
    - RoBERTa Tiny
    - MobileBERT
    - DistilBERT

The specific model for each module was selected by evaluating them on a 25% dataset that we split from then given training data and the one that gave the best performance was chosen. We found that very light models like MobileBERT and DistilBERT although improved the inference speed but did not perform well enough for the tasks.

2) Pre-Computing The Context Embeddings:

- We used a vector database for fast storage and retrieval of the context embeddings. We had precomputed these so that we did not have to do this while inference. The only embeddings calculated during the inference time were the new questions embedding. Doing this reduced our total inference time by about 5 seconds.

3) Parallelising on the CPU:

- The biggest speed up in our performance was from the use of the Ray Library. This allowed us to exploit the multiple cores of the CPU to parallelise and create n instances of our model (where n = number of CPU cores).

- Doing this reduced our average inference time by a factor of n. This is because the models were running parallely.

- Google Colab free tier offers upto 2 CPU cores so we were only able to half our inference time but our inference script can easily parallelise over how many ever CPU cores are provided leading to an astounding speed up.

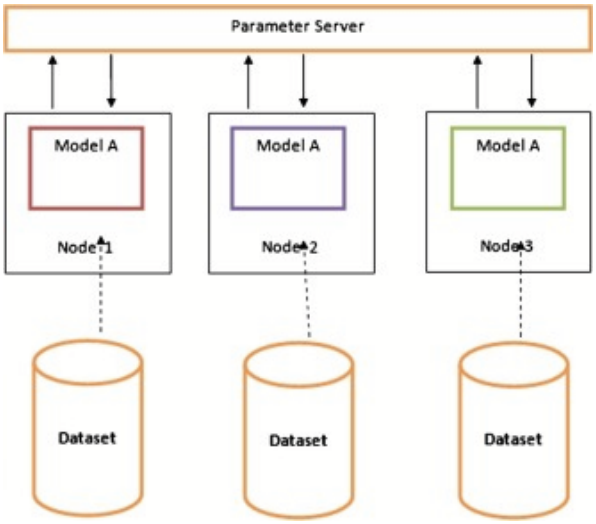

Fig 5. Diagram Showing Model Parallelism on CPU

Other Experiments and Ablation Studies:

1) Pseudo Question Generation and Data Augmentation:

- A major issue we had while training neural networks to improve the document embeddings was that there were quite a few unanswerable questions in a few themes and also that some passages did not have enough questions. This led to poor embedding formations and an overall decrease in performance.

- To tackle this we tried creating Pseudo Question Creation using generative models. This however took an extremely long time, approximately 1.5 hours to generate 10 questions for one passage. This method seemed unfeasible as 12 hours was far too less for data generation for all 30 themes.

- Another thing we tried was data augmentation of our existing dataset using NLP-Albumentations and NLP-Aug libraries. We used augmentations like:
    - Sentence Reordering
    - Synonym Replacement
    - Random Swaps
    - Random Deletions

2) Fine-Tuning on Each Theme and Making Theme Specific Models:

- We tried fine tuning the RoBERTa Tiny reader model on each theme to make it specialised on answering questions from a particular theme.
- While evaluation though we found out that the model started overfitting the training data and the score dropped off quickly
- The final score we got after the theme-wise training was only about 0.53.
- We believe that this is because the pre-trained models on the large SQuAD v2 dataset are more generalised and robust to varied themes.

Final Result and Conclusion:

Our proposed pipeline achieves a final metric of 0.74 for paragraph retrieval task and 0.71 for question-answering task. This is a huge improvement over our Mid-Eval results and we believe it can be further improved using a little more data and better hyperparameter tuning as mentioned in our proposed experiments.

The major drawback of the problem statement after the hardware constraints was the lack of data and inability to use external data. This hindered our training of embedding improval models.

Run time analysis of the Contrastive Learning Models:

- **Training Time for Each Theme:** 1 minute
- **Inference Time:** 0.001 seconds for 1 embedding
- **Model Size:** 12 Mb

Literature Review:

Most of our literature review was done before the mid evaluations. The only review we did after was regarding the contrastive learning objective and improving the word embeddings. These were all the papers:

1) Unsupervised Document Embedding via Contrastive Augmentation by *Luo et al.*
2) Adaptive Margin Diversity Regularizer for handling Data Imbalance in Zero-Shot SBIR by *Dutta et al.*
3) Towards Universal Paraphrastic Sentence Embeddings by *Wieting et al.*
4) SimCSE: Simple Contrastive Learning of Sentence Embeddings by *Gao et al.*

Final Pipeline:

Our final pipeline looks something like this:

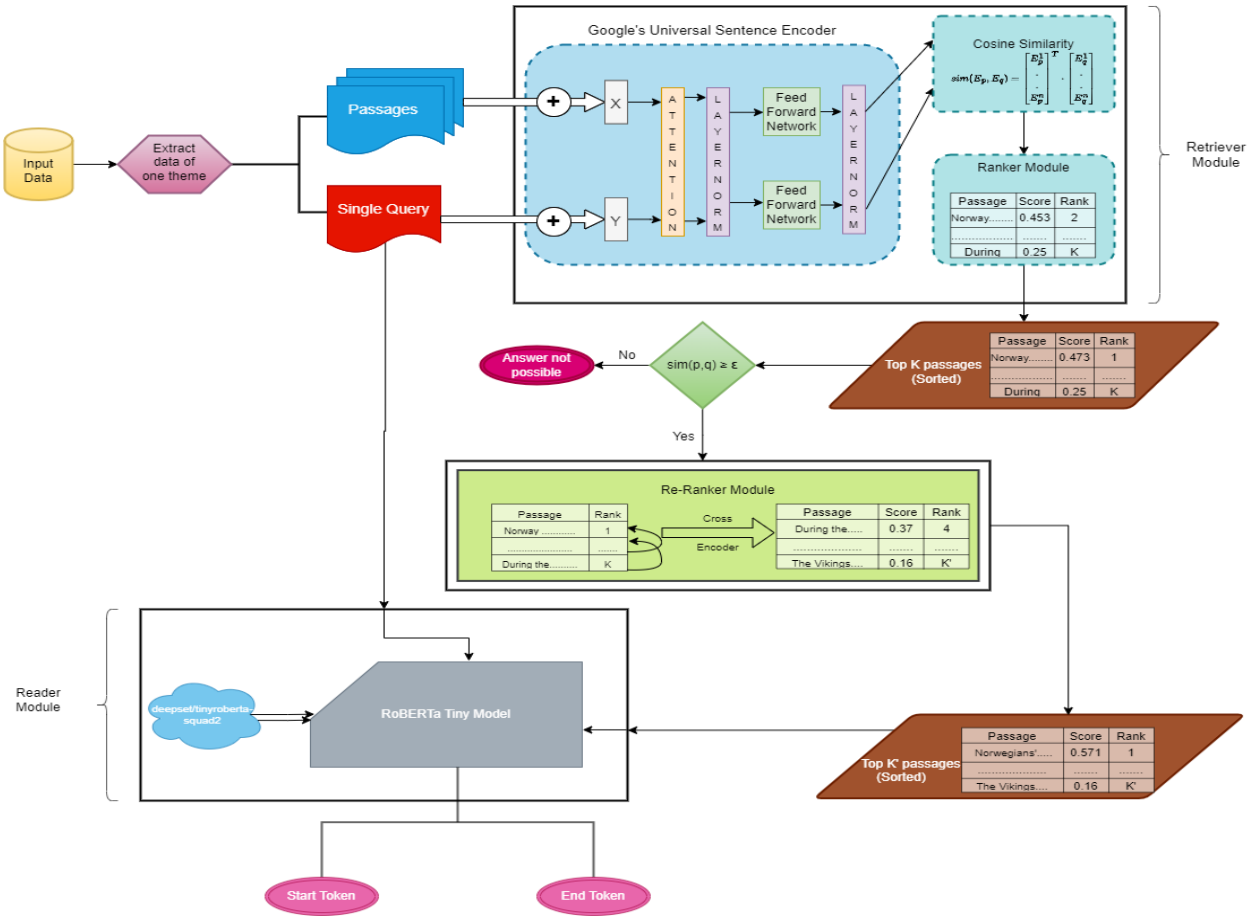

Fig 6. Final End-to-End Pipeline