Abstract

The report outlines our pipeline for domain-specific question answering in a open-QA setting. We have created **domain-adaptable rankers** fine-tuned using **knowledge distillation** in order to re-rank the passages retrieved using BM25. We propose a **novel difficulty prediction heuristic** which dynamically determines the number of paragraphs to be fed to the reader by utilising the **ranker scores and the remaining time**. Finally, we use **signals from reader, ranker as well as the retriever** to determine the answerability of the question.

Implementation

Paragraph Retrieval Task

Synthetic Data Generation

For training the ranker, we augment the dataset **10 times** by mining 10 **hard negative** paragraphs for each query using the BM25 retriever. This enables us to use k**nowledge distillation** as it is quite data hungry and also gives us the ability to use a **contrastive loss function** for training the ranker, details of both are outlined below.

Training methodology

A. Knowledge Distillation

The process involves a smaller student model trying to generalise and imitate a larger teacher model. The student model learns from the output logits of the teacher model on the dataset and also the ground truth values of the dataset.

Here we have used ms-marco-MiniLM-**L-12**-v2 cross-encoder as the teacher model to distil a pre-trained ms-marco-MiniLM-**L-2**-v2 cross-encoder as the student model. Both the models are pretrained on the ms-marco dataset for the task of **passage re-ranking**. We therefore **leverage task-transfer** by training it on the given dataset using knowledge distillation.

As mentioned before, for each query, we pair it with the 9 other hard negatives along with the one ground truth paragraph. For each query, we used the prediction logits from the teacher model as our **ground truth logits** for our distilled model and **minimised the mean square loss (MSE) between the two models.** The model was trained for 13 epochs on 80:20 train-test split with overlapping themes.

As this can also be modelled as a binary classification problem, we tried using Binary Cross Entropy Loss with the ground truth labels but we found that **MSE vastly outperforms BCE**.

	Top 1 Accuracy	**Inference Time per Query (Colab CPU)**
Student Model (MiniLM-L-2)	85.79	305 ms
Finetuned Model	89.31%	305 ms
Teacher Model (MiniLM-L-12)	90.27%	1010 ms

We can see that upon training the model with knowledge distillation we see an improvement of 9-10% compared to the pre-trained crossencoder. Thus **we use this as an universal ranker** along with **theme specific rankers** details of which are mentioned in the subsequent session.

B. Contrastive Loss function

To train the ranker, we also explored the use of **contrastive loss**, a loss function which **enables us to make the query closer to the positive paragraph than to the negatives in list of ranked paragraphs.** The loss function is mathematically written as

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \quad _{[k \neq i]} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)},$$

Here, we consider the negatives to be the hard negative paragraphs mined by the BM25 retriever in the data augmentation step. The **sim(z_i, z_j) is taken as the logits score of the cross encoder** on each of the 10 query, paragraph pair. One can notice that the value inside the log is essentially the softmax with respect to the positive query, paragraph pair. So, when we minimize this loss function, **we simultaneously maximize the logit score between the positive query, paragraph pair while minimizing the same for negative pairs.** However the improvement in the ranker results are not as pronounced as in the case of knowledge distillation as it can be seen in the table below.

Epochs	**Top 1 Accuracy**
Pretrained	81.02%
Epoch 1	81.65%
Epoch 2	82.71%

Fine-tuning Methodology

Upon fine tuning the rankers on specific themes using knowledge distillation method outlined above, we are able to bring improvement to the ranking accuracy compared to using a generic ranker trained on all themes. However, for themes that don't contain many training examples, we don’t see an appreciable difference in the top 1 accuracy and **thus we include only 2 theme specific rankers** in the final pipeline which are loaded at inference time. The table below shows the top 1 accuracy of the ranker for different themes in the testing data.

	Universal Ranker	**Theme Finetuning**
New York City	0.747	0.761
IPod	0.804	0.885
2008 Sichuan Earthquake	0.843	0.862

Run-time Improvements (Difficulty Heuristic)

For answering each question, we pass some of the top paragraphs from the retriever/ranker pair to the reader. We will then consider the reader confidence scores for each paragraph to fix a final paragraph where we expect the answer to lie (or we can even determine that answer doesn’t exist). We assume that the reader is smarter than the retriever/ranker pair, which is a fair assumption.

Now, we know that the **reader is much slower than the retriever (and ranker)**. The number of paragraphs passed to the reader is limited because of the time constraint. Let’s assume for the moment that we can pass two paragraphs to the reader, for each question. The simple strategy we can follow is to always pass the top two paragraphs.

We can optimize this strategy by noting:

1. The evaluation criteria limits us to use one second on average for each question in a theme. It doesn’t restrict us to predict the answer for each individual question in one second.
2. Not all questions are equally difficult. For some questions, the retriever/ranker pair could be fairly confident that the answer lies in a particular paragraph. For a particular question, the retriever could be confused regarding the existence of the answer in the top few paragraphs.

Let’s use these observations to efficiently allocate the constrained time to different questions. In our pipeline we find the top five paragraphs using the BM25 retriever and

rerank these top five paragraphs using our ranker model. Let $p(X,\ i)$ **denote ground truth probability that the answer to a question exists and lies among the top** i **of the ranker's final ranked paragraphs**, given that X is a concatenated vector of retriever/ranker scores. Let $q(X,\ i)$ **denote the ground truth probability that the reader will solve the question correctly**, given that the reader has been passed the top i paragraphs as ranked by the ranker, and X is a vector defined as in $p(X,\ i)$.

Let's say our model decides (in some way) to pass the top $z[i]$ paragraphs to the reader. What is the **expected number of questions correctly** solved by our entire model? It is:

$$\sum_i p(X_i,\ z[i])\ \cdot\ q(X_i,\ z[i])$$

Also note that the total time taken can be reasonably approximated as a linear function of summation $z[i]$, where k is some constant (since time taken by the reader to evaluate x paragraphs can be taken as a fixed linear function in x). Thus, we can arrive at an upper bound on summation $z[i]$ based on the time constraint (simply by a few trial runs). If we want to maximize expectation of the number of questions answered correctly, we have the following optimization problem:

Maximize expectation over the constraint- $\sum z[i]\ \leq\ K$ **for some** K**.**

This is a complicated optimization problem, so we have made the following simplifications:

1. Assume q is just a constant.
2. Use $\widehat{p}$, instead of p, which is a prediction generated by some model of ours.

Based on these simplifications, we need to do the following:

1. Come up with a model for $\widehat{p}$.
2. Come up with a heuristic solution for the optimization problem

Model for $\widehat{p}$

For $\widehat{p}$, we simply train a neural network with one hidden layer, with the X_i vectors as input the ground truth vectors y, where $y[i]\ =\ 1$ if answer exists and lies among the top i paragraphs of the ranker, in the train data and 0 otherwise.

Heuristic Algorithm

For optimizing, we can initialise all the $z[i]$ values by one. Then, for as long as $\sum z[i]\ <\ K$,

we keep incrementing the $z[j]$ **variable which locally increases the expectation by the maximum amount**. We can execute this process efficiently by using a min heap (heapq in python). Complexity is $O(K\ log\ n)$, where n is the number of questions. Note that this is just an approximate solution as we only focus on **local profits.**

Further we can follow this with a **random algorithm**, which repeats the following for a fixed number of times: **Randomly choose a j with $z[j] > 0$, decrement $z[j]$ and then again increase the $z[k]$ value which causes the maximum increase in expectation.**

We initially assumed that we pass two paragraphs to the reader for each question, but now we determine the value of the **constraint K dynamically based on time remaining**.

Evaluation Metric

We already split the data given into training and validation data, so we use the y vectors of the validation data (y are vectors for the validation set, where for each question, $y[j] = 1$ if answer for the question exists and is in the top j paragraphs or 0 otherwise. Our score will simply be **summation $\sum_i y_i[z[i]]$.** We can divide this with the number of questions with answers to find answering accuracy.

Evaluation

When using the simple method of **simply passing the top two paragraphs** for each question, we get an answering accuracy of 0.854. When **using the optimizer, we get an answering accuracy of 0.898 which is** quite considerable improvement as it increases the chance of the reader to correctly answer the question.

Using Already Answered Questions

We decide to use previously answered questions to determine which paragraph the answer may be found, in an attempt to assist our rankers. We set up a two-stage pipeline for this purpose. In the first stage, we use a sentence transformer model to encode all the questions (both current and previously answered) and compute their cosine similarity. We manually applied a 75% threshold to generate a first set of **similar questions**. In the second stage, we use a cross encoder model that has been specifically trained to detect similar questions finetuned on the **quora duplicate questions dataset.** If there are multiple similar questions for a given question, **we take the intersection of the top three answered paragraphs for all of them and use that as the answer**. However this involves keeping track of all the questions and paragraph used to answer it and iterating through it in runtime which can be a costly operation and thus it is omitted.

Implemented Pipeline (Stage 1)

The first stage of our pipeline tackles the paragraph retrieval with the use of an elastic search based BM25 retriever as well as a cross encoder ranker to re-rank the top 5 paragraphs. The retriever acts as a **fast yet effective low level filter** as it is able to capture the matching paragraph nearly 95.4% of the time in the top 5 while taking a maximum of 20ms per query.

We use **theme-specific as well as universal** cross encoder ranker **fine tuned using knowledge distillation** to compute the paragraph-query relevance using which we rerank the top 5 paragraphs fetched by BM25.

We then predict the probability of the passage being presenting in the top i in the updated ranking. We use this as well as the time left and apply a heuristic to determine the number of passages to be fed to the reader so as to **prioritize difficult question** which will require to examine more paragraphs for answers.

Question Answering Task

Synthetic Data Generation

To fine-tune the model, we propose a novel and unique type of data augmentation which involves pairing a query with a negative paragraph concatentated with only the **sentence in the positive paragraph that contains the answer**. The intuition behind this was that the reader model will learn to work with minimal context and thus might perform better. To **increase the hard negatives examples**, we pair some questions with a paragraph that doesn't contain the answer to that question but has a good reader score. This was done to improve the reader's ability to identify unanswerable questions. Our hypothesis is validated as outlined in the next section.

Training Methodology

As outlined in the table during the midterm report, minilm-uncased reader **offers the best performance-latency ratio.** Thus, we have used the pretrained weights of minilm-uncased for training on the given dataset. We have taken a preferred split of 80:20 on the whole training data mainly in three different ways:

1) By taking a **theme independent split** along the same ratio
2) By taking a **theme dependent split** of taking 80% in each theme for fine-tuning and the remaining 20% of each theme for validation purposes.
3) By taking 80:20 split on the augmented data using method mentioned above

Split type	**Details of fine-tuning**	**Exact match accuracy**
Theme Independent Split	Pre-trained Minilm	78.142%
Theme Independent Split	Minilm fine-tuned on the train-split	74.890%

Theme Dependent Split	Pre-trained Minilm	78.142%
Theme Dependent Split	Minilm fine-tuned on the train-split	75.217%
Data-augmentation	Minilm fine-tuned on the train-split	70.126%
Data-augmentation	2nd model fine-tuned again on the train-split	65.515%

On fine tuning the pre-trained model on the augmented data we found a exact match of 70.1% while using the earlier theme-wise fine-tuned model we found a exact match of 65.5% as show in the table.

Fine-tuning Methodology

Knowledge distillation

As many of the SOTA NLP models have a very high usage of storage and are computationally very expensive, we tried using **distillation in order to produce good results on much lighter models.** It involves a smaller student model trying to imitate a huge teacher model like BERT large. Hence, we have **distilled BERT-large to TinyBERT** with prediction and intermediate layer distillation. We leveraged **data augmentation** (explained below) to increase the number of training samples by a factor of 2. In reality we need to augment the data by a larger factor to have an effective student model but this makes the training very time consuming, hence was discarded.

Run time improvements

Decoding Strategy

From the model we have been using, we get outputs in the form of **start_logits and end_logits** for each example. We are to find the best possible answer using these probabilities, by checking various conditions like if the answer lies within the context or is the start index greater than the end index.
While writing the training and inference pipeline for readers, we designed three different types of **decoding strategies** -

1. **An efficient tensor decoding strategy** that uses the matrices. We reduce and prune the sample space to find the top n best answers by maximizing the sum of start_logits and end_logits while using the other conditions required for a feasible answer. It allows us to use **vectorization** and gives us a better time-optimal solution.
2. Another **novel solution of O(nlogn)** time complexity is using binary search and a type of sliding window of maximum possible answer length whose boundaries are found using binary search in O(log n) time complexity.

3. Lastly, the third decoding strategy, most **commonly implemented** is the simple searching algorithm of time complexity of $O(n^2)$ while only keeping the basic conditions in consideration.

However, since training didn't improve the reader by much, we used the pre-trained reader which uses matrix based approach for decoding the logits.

Implemented Pipeline (Stage 2)

Following the various experiments mentioned above, we found that fine tuning pre trained readers on the given dataset hurts its performance as it seems to **overfit** the data. Training the reader from scratch with the given dataset might bring improvements but we were **largely limited by its time constraint**. It was thus wise to use pre trained readers and upon careful examination of the latency/accuracy ratio, we used **minilm-uncased-squad2** which has 78.85% while taking only 305ms. This enables us to read on **average 2 paragraphs per question** which vastly increases the chances of finding the right answer.

To improve the answerability task, we propose a novel method in which we use the **confidence score of both the reader and the top 5 ranker scores to determine the answerability** of a question. The intuition is that the retriever scores must also be high if the reader's confidence is high. Upon experimentation, the hypothesis is validated, as we see about 2% increase in answerability classification with the extra signals.

Based on the answerability decided by the classifier, we either output the best answer determined by the reader or an empty string if it is not answerable.

Result & Runtime Analysis

The following table summarises the results for the theme dependent and theme independent splits (mentioned in training methodology) for the entire implemented pipeline.

	F1 Score	**Pipeline Inference Latency**
Theme Independent	72.79	0.973ms
Theme Dependent	74.12	

Conclusion

We were effectively able to tackle the problem of domain adaptability in the context of open question answering while maximising the latency and F1 score by using reader, retriever and ranker score signals and applying heuristics.