DevRev's Expert Answers in a Flash
Improving Domain-Specific QA

Team 45

Abstract

In this work, we develop pipelines to retrieve a knowledge base article from the database based on the query and answer the query using the retrieved passage. We optimize the pipeline for performance, latency, and resource usage. The availability of diverse knowledge bases makes this task challenging. We propose novel methods to handle FAQs, generate synthetic queries, model fine-tuning, retrieve candidate sentences for answer matching and improve runtime efficiency.

1. Our Pipeline Overview

Our final pipeline for the given task includes the following parts (Figure 1):

1. Previously Answered Questions (Section 2)
   - Fuzzy match with paraphrased questions
2. Sentence Retrieval (Section 3)
   - Retrieval of **Top K(dynamically chosen)** sentences to form the context for the question-answering model.
3. Question Answering (Section 4)
   - Question-Answering model inference on the retrieved context.

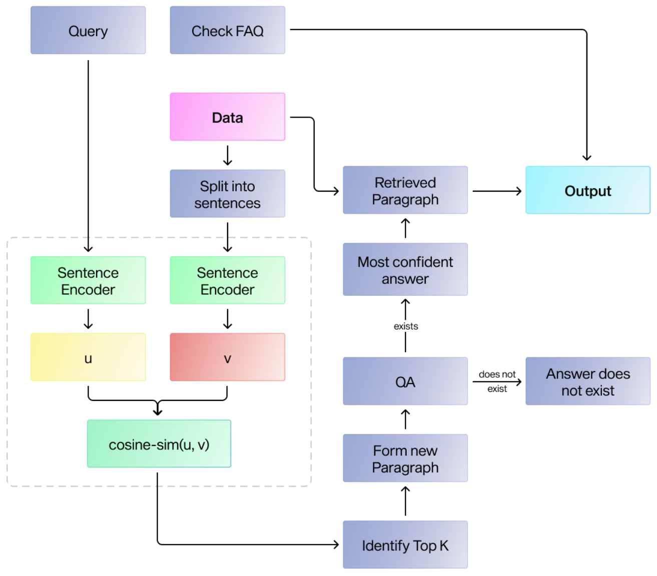

Figure 1. Final Merged Pipeline

2. Previously Answered Questions:

Current Approach: In this method, we used the answers to the available questions to respond the given query. To implement this, we paraphrased the questions of the given question-answer pairs using pre-trained T5-base paraphraser model[1]. We first check the input query with the questions in the paraphrased data using the FuzzyWuzzy[2] library and return the answer if the query exists in the generated paraphrased data otherwise, we forward this query to the Question Answering pipeline. We reduced the runtime on the CPU by converting this model to ONNX and then quantizing it using the fastT5 library.

We can make further improvements by establishing a trade-off between the amount of paraphrased data and the fuzzy matching threshold.

3. Paragraph Retrieval Task

3.1. Phrase v/s Passage Retrieval

Current Approach: Currently we are using **Sentence-based retrieval** in our final pipeline. We follow the intuition that a phrase/sentence based retrieval approach naturally entails retrieving larger passages. This approach is particularly appealing because the phrases/sentences can be used directly to extract answers. This approach significantly improves the retrieval accuracies and QA F1 scores.

Phrase based approaches don't take into account answers that span multiple sentences. Also, this approach can lead to possible context loss (context that is provided by other sentences around it), but context loss is also an issue for passage based approaches in the form of mean pooling.

Model Name	Top 1	Top 3	Top 5	Top 10	Inference Time
cross-encoder/ms-marco-TinyBERT-L-2-v2	0.79	0.92	0.95	0.98	660 ms
msmarco-distilbert-base-tas-b	0.73	0.88	0.94	0.98	35 ms
msmarco-distilbert-dot-v5	0.70	0.86	0.90	0.96	44 ms
multi-qa-mpnet-base-dot-v1	0.77	0.92	0.95	0.99	71 ms

Table 1. Bi-Encoders v/s Cross-Encoders Performance

3.2. Bi-Encoders v/s Cross-Encoders:

State of the art Cross-Encoders achieve better performance than Bi-Encoders. However, for most applications they are not practical due to large runtimes and inefficiency. They don't produce reusable embeddings which could be efficiently indexed and stored. Overall, our results indicate that the **cross-encoder performance boost is not significant enough to compensate for the large inference time**. Hence, we decided to move forward with the embedding-based approaches.

3.3. Synthetic Data Generation:

Question and answer generation is a data augmentation method that aims to improve question-answering (QA) models given the limited amount of human-labeled data. This improves model robustness on diverse and challenging datasets.

The following techniques are being used currently for synthetic data generation :

3.3.1. Answer Aware Question Generation:

In answer-aware question generation, we use a language model to generate a question based on the answer and its corresponding passage context. A pre-trained T5-small answer extraction model was used to retrieve entities from the passage, and these entities, with the paragraphs, were fed to a T5-base[1] model. We customized the pipeline to improve generation time and reduce RAM usage on the device. We achieved runtime reduction using fastT5 library and model quantization. We also observed huge spikes in RAM usage while processing larger passages. To mitigate this issue, we divided larger paragraphs into subparagraphs with shorter context lengths. Hence, we **used** this method for question answer generation.

3.3.2. NER Reranking:

While Generating Questions by Answer Aware QG technique, we faced the problems such as exceptionally high resource usage when the number of sentences increased. To filter the candidate sentences, we use a two-step approach as follows

[1]The reference of the question generation pipeline is given here

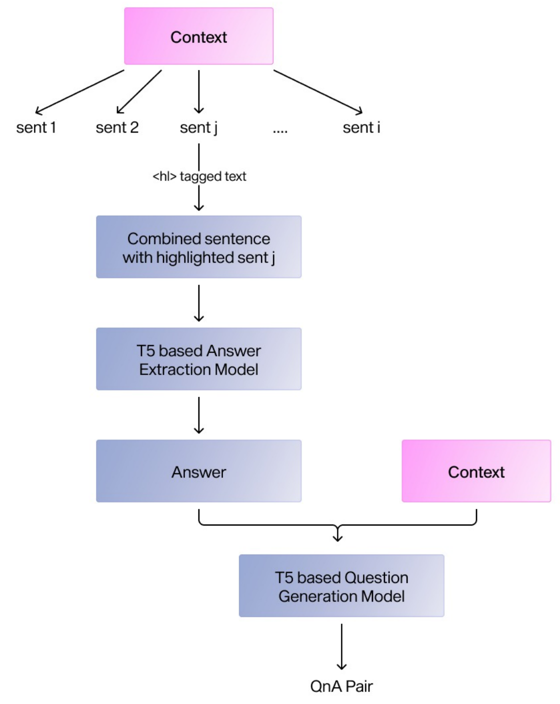
Context
sent 1
sent 2
sent j
....
sent i
<hl> tagged text
Combined sentence with highlighted sent j
T5 based Answer Extraction Model
Answer
Context
T5 based Question Generation Model
QnA Pair

Figure 2. Answer Aware QG Pipeline

- We passed the paragraph through a NER Model to identify key entities like people, organizations etc.
- We calculate the importance of each entity using **frequency, centrality and specificity** and then use a custom algorithm to re-rank the generated NER tags, thus generating only the most specific questions from the given theme.

In this method, we choose the best sentences instead of passing a random sample or passing all the sentences.

The following techniques were explored previously for synthetic data generation :

3.3.3. Rule-Based Question Answering:

The Stanford Parser, a Probabilistic Natural Language Parser, was utilized to analyze the paragraphs of a text.

The process involved dividing the paragraphs into segments, tokenizing them, and using NLTK, POS Tagger, and Parser to identify the Parts-of-speech and chunked output of each segment. The segments were then checked for clauses and based on specific rules defined in a referenced paper, the clauses were identified. Finally, other segments were appended to generate questions related to the original clause. Although this method was efficient due to its use of rule-based techniques, it was ultimately **rejected** because the generated questions were biased toward certain templates.

3.3.4. Question-Answer Based Semantic Role Labelling (QASRL)

The QASRL approach was employed to annotate question-answer pairs in a given paragraph. The technique models the verbal predicate-argument structure of the text using question-answer pairs. The parser initially detects unlabeled spans to determine the subject of a verb and then uses QG to label the relationship between the predicate and the detected spans. The model consisted of a pipeline in which verbal predicates were identified using POS tags, and unlabeled spans were detected to select a set of arguments for the verb. These spans were then grouped by questions to form a set of answers. However, this method was ultimately **rejected** because it could not be optimized for the best performance in Colab.

3.4. Training Methodology

Siamese networks are often used to extend sentence embeddings from English to other languages. We alter this approach to train sentence transformer for retrieval. Classically, we pass the English sentence embedding to a teacher model and train a student model to generate the same embedding for a translated sentence via Mean Squared Error (MSE) Loss. A secondary MSE Loss is calculated using English embeddings from the student model and teacher model, to ensure the student model maintains it's performance on purely English tasks as well.

Current Approach: To adapt the technique for retrieval task, we pass in the question to the teacher model which we do not train. The paragraph which answers the question is passed into the trainable student model, and MSE loss is calculated on these two embeddings. The aim is to train the student model to give similar embeddings for positive question and paragraph pairs, and maximize similarity for retrieval. Note, that we do not use negative question paragraph pairs for this task. We also use the same model for teacher and student, as our purpose is to improve the embeddings. We implement our technique on two sentence-transformer models[6] and notice significant improvements in retrieval.

3.5. Domain Adaption:

Domain Adaptation is a technique to improve the performance of a model on a target domain containing insufficient annotated data by using the knowledge learned by the model from another related domain with adequate labelled data.

Retriever: We use dense retrievers for the retriever task, which are known to degrade in performance across unknown domains. We explored the novel unsupervised technique, Generated Pseudo Labelling(GPL), to adapt the retriever for the new domain. The technique involves three steps: First, synthetic queries are generated for each passage from the target corpus using synthetic question generation techniques(reference section). Then, the generated queries are used to mine negative passages with the use of dense retrievers. Finally, the query-passage pairs are labelled by a cross-encoder and used to train the domain-adapted dense retriever. The method has shown state-of-the-art performance on various QA datasets such as FiQA, SciFact, BioASQ, and Robust04. However, we do not use this technique as it requires time, and all the steps are resource intensive, which cannot be met with the given constraints.

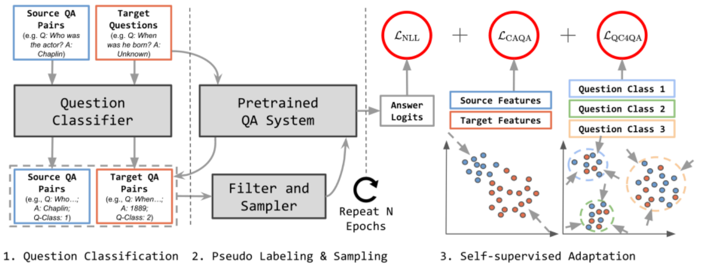
Source QA Pairs (e.g. Q: Who was the actor? A: Chaplin)
Target Questions (e.g. Q: When was he born? A: Unknown)
Question Classifier
Source QA Pairs (e.g., Q: Who...; A: Chaplin; Q-Class: 1)
Target QA Pairs (e.g., Q: When...; A: 1889; Q-Class: 2)
Pretrained QA System
Filter and Sampler
Repeat N Epochs
Answer Logits
$\mathcal{L}_{\text{NLL}}$
+
$\mathcal{L}_{\text{CAQA}}$
+
$\mathcal{L}_{\text{QC4QA}}$
Source Features
Target Features
Question Class 1
Question Class 2
Question Class 3
1. Question Classification
2. Pseudo Labeling & Sampling
3. Self-supervised Adaptation

Figure 3. Overview of QC4QA method

Reader: Once deployed, QA systems often experience performance deterioration on user-generated questions. Such performance drops can be traced back to domain shifts in two input elements: User-generated questions are syntactically more diverse and, thus, different from the training QA pairs The context domain of test-time input (target domain) can oftentimes diverge from the training corpora (source domain), e.g., from news snippets to biomedical articles. We explored the Question Classification for Question Answering (QC4QA) technique for improving reader(QA model) performance on QA tasks. QC4QA can be divided into three stages: Starting with Question classification, where all questions are assigned to different classes. Then Pseudo labelling and sampling, where we label and sample target examples with the distribution-aware sampling strategy. And finally, Self-supervised adaptation, in which we train the QA system jointly with source and target data. This method has shown state-of-the-art results with SQUAD as a source

Model Name	Top 1 sentence	Top 5 sentence	Top 15 sentence	Top 20 sentence
msmarco-distilbert-base-tas-b trained	**0.679**	**0.884**	0.934	0.946
msmarco-distilbert-base-tas-b	0.640	0.851	0.922	0.937
multi-qa-mpnet-base-dot-v1 trained	0.669	0.880	**0.948**	**0.958**
multi-qa-mpnet-base-dot-v1	0.617	0.849	0.932	0.946

Table 2. Siamese Trained Models vs Pre-Trained Models on Retrieval Accuracy

domain on various target domains such as triviaQA, CoQA, NewsQA etc. However, we do not use this technique as it requires time and resource-intensive self-supervised adaptation, which cannot be met with the given constraints.

3.6. Fine-Tuning Methodology:

Experimental Setup: We fine-tune our bi-encoders/embedding models that are pre-trained on the MS MARCO dataset for the retrieval task using Multiple Negatives Ranking Loss. For fine-tuning, we use the synthetic data that we have generated along with the provided question-answer pairs. We fine-tuned across three levels of specificity: global, cluster, and local. Across all settings, we fine-tuned for 1 epoch with batch size 8, warm-up steps equivalent to 10% of the generated data. To use MNR loss, a custom dataloader was implemented to that there are no duplicate sentences within the same batch.

Global fine-tuning involved using the entire generated data, while cluster-based fine-tuning used only a subset of data (based on clustering of themes) and single theme fine-tuning created models for each theme by fine-tuning on each theme separately. The clustering of themes was done by creating Phrase-Bert-based embeddings for the theme names and then applying a clustering algorithm like KMeans, Affinity Propagation etc. The intuition behind clustering based on theme names is that themes with similar knowledge would be in the same cluster.

Why MNR Loss? Multiple Negative Ranking (MNR) Loss is a suitable loss function because we have only positive query, passage pairs. When we use MNR loss, we provide triplets: (query, positive_passage, negative_passage) where positive_passage is the relevant passage to the query and negative_passage is a non-relevant passage to the query. We compute the embeddings for all queries, positive passages, and negative passages in the corpus and then optimize the following objective: We want to have the (query, positive_passage) pair to be close in the vector space while (query, negative_passage) should be distant in vector space. We don't actually provide the negative_passages, they are generated in batch. For each positive pair of (query, passage) that can be represented as (qi, pi), we form negatives in the form of (qi, pj) where $i! = j$.

An alternative to MNR loss is MarginMSE loss. An advantage of MarginMSELoss compared to MNR loss is that we do not require a positive and negative passage. In contrast to MNR loss which uses the (query, passage1, passage2) triplets, passage1 and passage2 do not have to be strictly positive/negative, both can be relevant or not relevant for a given query. We then compute the Cross-Encoder score for (query, passage1) and (query, passage2). We then compute the distance:

$$CE\_distance = CEScore(query, passage1) - CEScore(query, passage2).$$

For our bi-encoder training, we encode query, passage1, and passage2 into vector spaces and then measure the dot-product between (query, passage1) and (query, passage2). Again, we measure the distance:

$$BE\_distance = DotScore(query, passage1) - DotScore(query, passage2).$$

We then want to ensure that the distance predicted by the bi-encoder is close to the distance predicted by the cross-encoder, i.e., we optimize the mean-squared error (MSE) between CE_distance and BE_distance.

A major disadvantage of MarginMSELoss is the slower training time. In MNR Loss, with a batch size of 32, we compare one query against 64 (32+32) passages while with MarginMSE Loss, we compare a query only against two passages which leads to slower convergence. Also the usage of cross-encoders further slows down the whole process.

Results: Fine tuning improves retrieval accuracy on direct passage retrieval task along with greater improvement on both retrieval and QA task in final pipeline. Results are referred in Table 3.

Fine Tuning type	Top1	Top5	Top10
Vanilla	0.64	0.87	0.93
Global	**0.72**	0.90	0.95
Local	0.69	**0.93**	**0.96**
Clusters	0.71	0.92	0.95

Table 3. Retrieval Accuracies for different fine tuning methods

3.7. Run Time Improvements:

Stored embeddings: We store the embeddings of the paragraphs in memory to save time on repeated calls to the model.
Dynamic sentence count: Dynamically choose a number of sentences based on a token threshold in the Reader-Guided Passage Reranking technique to make the inference time dataset independent and maximize score within constraints.

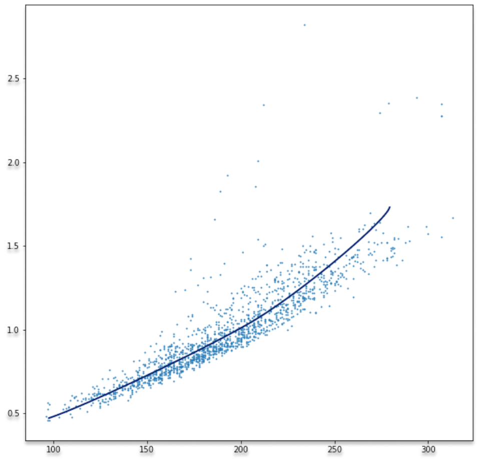

Figure 4. Token Length v/s Time

Quantization and Optimisation: We quantized and optimised(onnx) the sentence embedding models. However, the advantage in speed did not outweigh the loss in scores. Hence, we choose **not to use** this method.

4. QA Task

4.1. Synthetic Data Generation:

Details about this task can be found in **Section 3.3.1**

4.2. Training Methodology:

The given training data was a subset of the SQuAD 2.0 training data. Models trained on complete SQuAD 2.0 training data are available on Huggingface which perform well on the SQuAD 2.0 validation dataset. However, there are more QA datasets available (Trivia-QA, News-QA, Natural Questions, HotPot-QA) that have different distributions of questions and paragraph types due to which the SQuAD 2.0 models perform poorly on the same. In order to include information of another dataset (other than SQuAD 2.0) in our model we tried Model Distillation. The teacher model used for distillation was trained on a Natural Questions (**deepset/roberta-base-squad2-nq**), and the student model picked was trained on SQuAD 2.0 (**deepset/tinyroberta-squad2**). Due to the dataset constraints, using the given data for distillation caused the model to overfit on SQuAD 2.0. **We didn't go ahead with this approach due to the above reasons.**

Dataset	F1	F1-distilled
SQuAD 2.0 Valid	**82**	80
Trivia QA	43	**44**

Table 4. F1 score before and after distilling tinyroberta

Possible Solution: If we could use any dataset other than SQuAD 2.0 to distill the model using a teacher model, then it can help make the model more robust.

4.3. Fine-Tuning Methodology:

Experimental Setup: For fine-tuning questions and answering tasks, we used theme-wise and cluster-wise fine-tuning. We use the squad2 validation set for all our tests. We split the dataset theme/cluster-wise into train test pairs(30/70). On each theme/cluster, we fine-tune the combined dataset of the train(30%) and the generated questions (explained in 3.3.1) and test(70%) data. We use Phrase-Bert encoder to generate embeddings for theme titles and cluster them using KMeans Clustering.

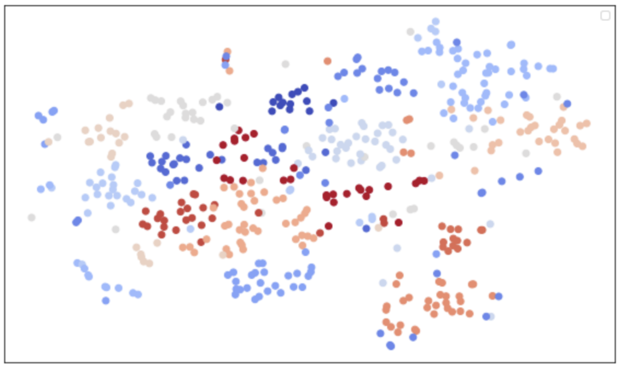

Figure 5. KMeans Clusters on theme embeddings

We notice that the themes with similar semantic information are clustered together (Fig 5). For example, the sky blue cluster in the lower-left part of the figure consists of all *football clubs* in the dataset, and the dark blue in the lower-center part of the figure consists of all *US cities*.

Results: We find that in both cluster and theme-based fine-tuned models, the F1 and Exact Match scores increase compared to the original pre-trained model on the test set. We also find that the variance of scores across themes reduces significantly. Thus, we preferred this idea for theme/cluster-based fine-tuned models. The cluster-based fine-tuning performs better than theme-based fine-tuning because the availability of data is more for a cluster than a theme.

Name	F1-mean	F1-van-mean
Cluster-Wise fine-tuning	**87.2**	86.8
Theme-Wise fine-tuning	**86.8**	86.4

Table 5. mean of F1 in model using fine-tuning and vanilla, where the number of clusters is 10

Name	F1-var	F1-van-var
Cluster-Wise fine-tuning	**6.9**	8.6
Theme-Wise fine-tuning	**31.1**	34.3

Table 6. variance of F1 in the model using fine-tuning and vanilla, where the number of clusters is 10

However, the above finetuning recipe on the generated **QA pairs on the training dataset** did not improve our scores. This shows that fine-tuning the generated QA pairs on the pre-trained paragraphs leads to overfitting and should be avoided. Further on, in the given test set, most of the QA pairs were also part of the squad2 train dataset on which the model was pre-trained. Hence, we decided against fine-tuning models on any theme or cluster.

4.4. Run Time Improvements

In order to improve inference time on the CPU of the Question Answering Model, we have tried the following approaches:

ONNX Representation and Model Quantization: ONNX[3] is an open format built to represent machine learning models. ONNX defines a common set of operators - the building blocks of machine learning and deep learning models - and a common file format to enable AI developers to use models with a variety of frameworks, tools, runtimes, and compilers We had the following latency results[2] using ONNX representation :

Deberta-V3-Base Model	Latency (per query)
ONNX Base	725.75ms
ONNX Optimized	669.62ms
ONNX Optimized+Quantized	**445.97ms**

Table 7. ONNX Model Inference Times on Google Colab

The **Quantized** here refers to **INT8** quantization of model weights **(dynamic quantization)**. We can see that the latency is minimum for the quantized model. However, the results produced by the quantized model were not satisfactory. The results for the quantized Deberta model were not accurate due to partial/improper quantization (not supported fully yet). The solution for this is using **Quantization Aware Training** (introduces fake quantization of inputs) during the fine-tuning of the models for a downstream task so that the models can learn to improve accuracy in INT8 precision. However, as we had dataset and training time constraints implementing this was not feasible.

[2]all latency results were tested on AMD EPYC 7B12 CPU Session. The latency was tested by averaging the model inference time on 20 random inputs (240 sequence length) in a for loop

Finally, we decided to go forward only with the Optimized ONNX model for the final pipeline. In order to get inference times further under the constraints, we limited the total number of tokens returned by the retrieval model, which caused a major boost in inference time with minimal effect on the F1 score(reference to retrieval section).

The F1 scores and latency details on SQuAD 2.0 validation dataset are in Table 8.

Model	F1	Latency (per query)
ONNX Optimized	85.48	**669.62ms**
PyTorch Vanilla	87.06	1112.73ms

Table 8. ***deepset/deberta-v3-base*** trained on SQuAD 2.0

ONNX models score a bit less compared to the vanilla PyTorch model. While ONNX models may score slightly lower than their PyTorch counterparts, the trade-off in terms of deployment flexibility, collaboration, and performance makes it a compelling option for many use cases.

Runtime Environment Challenges: Google Colab provides 2 runtimes in its free tier version; one with an *AMD EPYC 7B12* CPU and the other with an *Intel(R) Xeon(R) CPU @ 2.20GHz*. The inference times were as follows:

We can see a clear performance difference between Intel and AMD CPUs provided by Google Colab.

Possible solutions include using Intel-specific optimizations for inferencing ONNX models faster on Intel, like using Intel Integrated Graphics of the CPU and using FP16 or BF16 model formats for inference. However, none of the above methods is supported on an Intel CPU session available in free tier Google Colab.

Rust Implementation of Transformers: Rust is a lower-level language compared to Python, which makes it very fast. Like python, it also has libraries in the form of crates, and rust-bert[4] is one such crate that implements almost all the transformer models in Rust for inference.

One of the major advantages of using Rust is faster inference times and efficient memory usage without compromising on model accuracy. Rust models are highly optimized for Rust-based runtimes and require heavy installations and compilations to function effectively in a python based environment. This did not allow us to use the Rust models for submission. For industrial applications, where a custom environment is used, we propose to use Rust models for fast runtimes and optimized usage.

Model Pruning: Pruning is a systematic way of removing redundant weights and connections within a neural network. An applied pruning algorithm must determine which weights are redundant and will not affect the accuracy, which basically involves sparsifying the model weights to increase inference speed. For the same, we used SparseML[4], which is an open-source library that simplifies the process of applying sparsity algorithms. However, one of the major drawbacks of the above method is that it requires training the model from scratch with sparsification algorithms applied for the required downstream task. The training requires around 30-35 epochs which was not a feasible option according to the given constraints.

We also tried layer-drop pruning, wherein we remove a number of top encoding layers from a trained model, and re-train with layer-wise learning rate decay (LLRD). Essentially, the new top layer is assigned the highest rate, and this decays as we go down the layers. We show results for a 10-layer roberta-base model (original model has 12 layers) trained on SQuAD 2.0, which we re-trained for 1 epoch on the given training set. Recovering original performance needs more epochs which is not feasible given the constraints, and also runs the risk of over-fitting. The F1 scores for Table 9 are on SQuAD 2.0 validation set.

Model	F1
deepset/roberta-base-squad2 pruned	82.07
deepset/roberta-base-squad2	82.91

Table 9. Effect of layer-drop pruning + LLRD

5. Reader-Guided Passage Reranking:

The given passage corpus for the theme is converted into a sentence corpus using the NLTK sentence tokenizer.
The top k(parameter) closest sentences in the corpus to the query are retrieved using the Sentence Embedding models by cosine measure . These sentences are then concatenated and passed to the QA model as a single context.
If the QA model returns an answer, we retrieve the sentence which returned the answer. The sentence is then used to retrieve the actual paragraph it belonged to, hence completing the retrieval.
If the QA model does not return an answer, we say that the question is unanswerable and return -1 for the retrieved passage.
Since the QA model performs inference on the top k sentences, the inference time is directly proportional to the sentence length. This is a parameter we cannot control, we add an additional threshold on the number of tokens passed to the QA model. Hence, making our model's runtime dataset independent.

6. Final Implemented Merged Pipeline:

In a realistic setting, the given data would be unseen for both the Retrieval(Biencoders/Sentence Embedding) Models and Question Answering Models. Our approach is:
We fine-tune the Retrieval model theme-wise on a corpus comprising of any sample QA pairs present on the given theme and Generated QA on the paragraphs of that theme using MNR loss .
We fine-tune the QA model theme-wise on a corpus comprising of any sample QA pairs present on the given theme and Generated QA on the paragraphs of that theme.
We first check for FAQs and then follow it with the Reader Guided passage reranking technique using an optimised version of the Deberta model.

However, this was **not** the case for most of the test samples provided. We modified our approach to suit the data better. We did not fine-tune the QA model and only used an optimised version of the vanilla deepset Deberta model.

7. Result and Run-time Analysis

We get the following results on the Final Implemented Merged Pipeline:

Model Name	Retrieval	QA
van Ret + van QA	89.9	82.1
tuned Ret + van QA	90.3	83.02
van Ret + tuned QA	90.1	82.5
tuned Ret + tuned QA	91.45	83.77

Table 10. Pipeline Scores

– van Ret - Vanilla "all-mpnet-base-v2"
– tuned Ret - van Ret fine tuned locally
– van QA - Vanilla "deepset/deberta-v3-base-squad2"
– tuned QA - van QA fine tuned locally

Model Name	Retrieval	QA	Inf Time
van Ret, QA top 10	**89.9**	**82.1**	1.02
van Ret, QA top 5	87.91	80.03	605

Table 11. Time v/s Score tradeoff with no. of sentences

8. Literature Survey

Retrieval: Open Domain Question Answering (ODOA) has been studied widely recently, and a classic framework of ODQA system is implemented by encompassing an information retriever (IR) and a reader, i.e., Retriever-Reader. The task of IR is to retrieve evidence-related text pieces

from the large knowledge corpus. Popularly used I can be TF-IDF, BM25 and DR (dense passage retriever), etc. The target of the reader is understanding and reasoning the retrieved evidence to yield the answer. It is often achieved by transformer-based language models, such as BERT, RoBERTa, ALBERT or sequence-to-sequence generator T5, BART, GPT, etc. There is a very recent approach where we are stacking the retriever layer, reranking layer and reading layer into a single model name YONO which aims at reducing the model size.
There are different types of frameworks along which the question-answering models have been developed, namely:

- Retriever and Reader
- Retriever-only
- Generator-only

Most recent works follow Retriever and Reader framework and further supersede the TF-IDF or CNN based retriever with stronger transformer-based models, such as BERT, T5, BART, etc. The readers can be classified into generative and extractive readers. Dense Passage Retriever directly leverages pre-trained BERT models to build a dual-encoder retriever without additional pre-training. Dual-encoder retrievers like DR, encode the questions and documents independently, ignoring interaction between questions and documents, and limiting their retrieval performance. To remedy this issue, Colbert adds interaction between different embeddings on the top of a dual-encoder, and Colbert@A applies it into ODQA domain to gain better performance. Retriever-only systems tackle ODOA tasks with a single retriever, eliminating reading or generating step. Generator-only ODQA models are normally based on single generators, mainly seq2seq generative language models, like T5, GPT and BART. However, most general -purpose ODQA models are computationally intensive, inference slowly, and training expensive. One reason is the huge index/document size (see Table 2). Concretely, a corpus typically contains millions of long-form articles that need to be encoded and indexed for evidence retrieval. As we want our model to deliver answers quickly by using limited resources, all these resource-heavy and slow inference methods are not appropriate for our tasks.

Question Answering Early approaches to Question Answering involved rule-based systems, relying on predefined rules and patterns to extract answers from the text. These systems had limited capability to handle complex and ambiguous questions. They were not robust to out-of-domain tasks and languages. With the advent of Machine learning, large text datasets were used to train language models with a self-supervised training mechanism, leading to considerable improvements in answer extraction and generation. These methods relied on information retrieval (IR) techniques that involved ranking and retrieving the most relevant documents or passages from a given dataset. Recently, Large deep-learning models have offered significant accuracy gains, but training these large language models is challenging. These models are not usable in real-time applications due to resource and device constraints. Currently, many pre-trained models are available on Hugging Face (RoBERTa, BERT, Deberta-v3, XLM Net, etc.) for extractive Question Answering. Given the resource constraints in the problem statement, these models are rendered unusable for this task. We research possible methods to optimize the available models on CPU are: • Quantization • Pruning • Distillation • Model Architecture Most works on model compression focus on ”distilling” a pre-trained model through expensive finetuning, while some reduce model complexity by structured pruning of model parameters. Structured Pruning of BERT-based Question Answering Models uses a hybrid combination of task-specific structured pruning and distillation and shows significant gains in speed and performance. Zero Redundancy Optimizer (ZeRO) optimizes language model memory requirements, enabling lower latency.

9. References

[1] Mao, Yuning, et al. Rider: Reader-Guided Passage Reranking for Open-Domain Question Answering. arXiv preprint arXiv:2101.00294 (2021).

[2] Thakur, Nandan, et al. BEIR: A heterogenous benchmark for zero-shot evaluation of information retrieval models. arXiv preprint arXiv:2104.08663 (2021).

[3] Hazen, Timothy J., Shehzaad Dhuliawala, and Daniel Boies. Towards domain adaptation from limited data for question answering using deep neural networks. arXiv preprint arXiv:1911.02655 (2019).

[4] Fisch, Adam, et al. MRQA 2019 shared task: Evaluating generalization in reading comprehension. arXiv preprint arXiv:1910.09753 (2019).

[5] Henderson, Matthew, et al. Efficient natural language response suggestion for smart reply. arXiv preprint arXiv:1705.00652 (2017).

[6] Reimers, Nils, and Iryna Gurevych. Making monolingual sentence embeddings multilingual using knowledge distillation. arXiv preprint arXiv:2004.09813 (2020).

[7] Nils Reimers and Iryna Gurevych, Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks, 2019.

[8] Chen, D., Fisch, A., Weston, J., Bordes, A. (2017). Reading wikipedia to answer open-domain questions. arXiv preprint arXiv:1704.00051

[9] Raffel, Colin, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. The Journal of Machine Learning Research 21.1 (2020): 5485-5551.

[10] Wang, Shufan, Laure Thompson, and Mohit Iyyer. Phrase-bert: Improved phrase embeddings from bert with an application to corpus exploration. arXiv preprint arXiv:2109.06304 (2021).

[11] Nguyen, Tri, et al. MS MARCO: A human generated machine reading comprehension dataset. choice 2640 (2016): 660.

[12] Wang, Xinshao, et al. Ranked list loss for deep metric learning. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.

[13] Hofstätter, Sebastian, et al. Improving efficient neural ranking models with cross-architecture knowledge distillation. arXiv preprint arXiv:2010.02666 (2020).

[14] Mao, Y., He, P., Liu, X., Shen, Y., Gao, J., Han, J., Chen, W. (2020). Generation-augmented retrieval for open-domain question answering. arXiv preprint arXiv:2009.08553.

[15] Devlin, Jacob, et al. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018).

[16] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692.

[17] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., Soricut, R. (2019). Albert: A lite bert for self-supervised learning of language representations. arXiv preprint arXiv:1909.11942.

[18] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res., 21(140), 1-67.

[19] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., ... Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. arXiv preprint arXiv:1910.13461.

[20] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... Amodei, D. (2020). Language models are few-shot learners. Advances in neural information processing systems, 33, 1877-1901.

[21] Lee, H., Kedia, A., Lee, J., Paranjape, A., Manning, C. D., Woo, K. G. (2021). You only need one model for open-domain question answering. arXiv preprint arXiv:2112.07381.

[22] Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., ... Yih, W. T. (2020). Dense passage retrieval for open-domain question answering. arXiv preprint arXiv:2004.04906.

[23] Humeau, S., Shuster, K., Lachaux, M. A., Weston, J. (2019). Poly-encoders: Transformer architectures and pre-training strategies for fast and accurate multi-sentence scoring. arXiv preprint arXiv:1905.01969.

[24] Khattab, O., Potts, C., Zaharia, M. (2021). Relevance-guided supervision for openqa with colbert. Transactions of the Association for Computational Linguistics, 9, 929-944.

[25] Lu, Y., Liu, Y., Liu, J., Shi, Y., Huang, Z., Sun, S. F. Y., ... Wang, H. (2022). ERNIE-Search: Bridging Cross-Encoder with Dual-Encoder via Self On-the-fly Distillation for Dense Passage Retrieval. arXiv preprint arXiv:2205.09153.

[26] Khattab, O., Zaharia, M. (2020, July). Colbert: Efficient and effective passage search via contextualized late interaction over bert. In Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval (pp. 39-48).

[27] Mao, Y., He, P., Liu, X., Shen, Y., Gao, J., Han, J., Chen, W. (2021). Rider: Reader-Guided Passage Reranking for Open-Domain Question Answering. arXiv preprint arXiv:2101.00294.

[28] Reimers, N., Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084.

[29] Wang, K., Reimers, N., Gurevych, I. (2021). Tsdae: Using transformer-based sequential denoising auto-encoder for unsupervised sentence embedding learning. arXiv preprint arXiv:2104.06979.

[30] Gao, T., Yao, X., Chen, D. (2021). Simcse: Simple contrastive learning of sentence embeddings. arXiv preprint arXiv:2104.08821.

[31] Janson, S., Gogoulou, E., Ylipää, E., Cuba Gyllensten, A., Sahlgren, M. (2021). Semantic re-tuning with contrastive tension. In International Conference on Learning Representations, 2021.