THE SWIFT SWEEP

QUICK COMPREHENSION FOR THE WIN

GROUP ID: 25

INTRODUCTION

The project aimed to build an efficient and easy to deploy QA system that can handle answerable queries quickly and connect users with a human worker if necessary. A combination of efficient modules was chosen for paragraph retrieval and answering questions.Our implementation uses various Ranking and Reranking algorithms and modules to retrieve the top set of paragraphs. The Retrospective Reader model was selected for its human-like nature and efficiency in solving MRC problems. The implementation was optimized for inference time and memory usage, making it cost-effective for real-world deployment.

MODEL COMPONENTS

Synthetic Data Generation

With our new pipeline for generating synthetic data, we were able to create huge amounts of data within a considerably short span of time. (See Results). The method that we focus upon is Data Augmentation. This is more suited for fast paced Question Answer generation as compared to GANs.

Architecture

01 **Breakdown Mechanism:**
The paragraphs provided are broken down into small sentences and then finally into words. This chronological order thus gives a Single Question Per Line Approach.

02 **Stopword Filtering:**
Stopwords from the extracted corpus are filtered out using a NLTK corpus library.

03 **Vectorization:**
The corpus is then passed through a Sentence Transformer library which then vectorizes them.

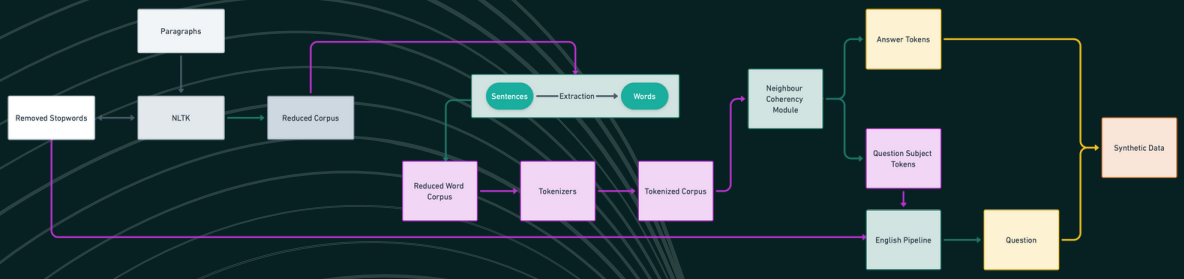
04 **Neighbour Coherency Module (NCM):**
The filtered vectors are then passed into NCM. NCM is essentially a simple grouping mechanism that works on a single vector and tries to associate it to its neighboring vectors.

05 **T5 transformer tokenization:**
These vectors then filter down to a subject vector and an answer token vector using Probability Clustering, a mechanism that uses a pre-built corpus to assign weights to each sequence of vectors.

06 **T5 conditional:**
The corpus of stop words is now traversed through in a sorted order and appended to the subject and answer tokens to get synthetically generated questions. This happens using a pre-trained T5 condition based architecture.

07 **Filtering out:**
To filter out meaningless questions a top-p nucleus sampling method is used that compares the answer and the subject tokens semantically and ranks them in descending order of similarity.

Visualization of Pipeline



BM25 Ranking Module

BM25 is a ranking algorithm which is non binary in nature and based on TF-IDF. For simple retrieval tasks, where we want to filter out irrelevant documents, BM25 provides the best efficiency and accuracy. For instance, as tested by our team, retrieving top 10 documents by BM25 provides an accuracy of 94.5% within a timespan of less than 10 ms. Method of application of BM25:

a. All paragraphs of a given theme are tokenized to form a corpus of subwords.

b. After obtaining the corpus, for every question, the BM25 algorithm is applied for the query to obtain the paragraph among the corpus which are most relevant. (See Appendix)

Cross Encoder

The CE Reranking Module re-ranks paragraph/information retrieval results. The architecture is essentially a neural net consisting of a pre-trained MiniLM Model consisting of six layers. This neural net is trained on MS Marco Dataset. The methodology of the algorithm consists of two phases:

a. Pre-processing: The preprocessing phase includes gathering information about keywords in the search queries, the content of the paragraphs, and the overall relevance of the paragraphs.

b. Re-evaluation: During the re-ranking phase, the algorithm uses the information gathered to generate a score for each outcome. This score is used to determine the final ranking of results. The algorithm also uses a set of weights to adjust the relevance of results. By using query context and content quality, algorithms can generate more relevant results for queries.

Retrospective Reader

This component implements a Retrospective Reader for MRC (using Electra as the PrLM base). It has 3 main subcomponents:

Sketchy Reader

This component implements a Retrospective Reader for MRC (using Electra as the PrLM base). Cross entropy loss function is used for training objectives. (See Appendix). It has 3 main subcomponents:

Embedding

First, the question and passage are concatenated as input text and then a tokenizer specific to an encoder is used to obtain tokens. Let T = {t1, t2, ..., tn-1, tn} represent a sequence of tokens of length n. Input embedding for each token consists of token embedding, position embedding, and token type embedding. Let the outputs of the encoder be denoted by X = {x1, x2, ..., xn-1, xn}. Finally, input embeddings are passed to the interaction layer to obtain the contextual representation.

→

Interaction

Encoder output X is passed to a multilayer transformer where it learns the contextual representation. This contextual representation is denoted by H = {h1, h2, ..., hn-1, hn} representing the last hidden state of the input sequence.

→

External Front Verification

It is implemented to identify unanswerable questions. The pooled h1 ∈ H is fed to a fully connected layer and classification logits are obtained, logitans for answerable and logitna for no answer.

Intensive Reader

The intensive reader uses the sketchy reader's same encoding and interaction mechanism to obtain contextual representation H and finally verify the answerability of the question.

Question aware matching

First, we split vector H into HQ and HP representing each passage and question respectively, according to its positional information. Then we employ a question-aware mechanism called multi-head cross-attention. Finally, question-aware contextual representation H` is obtained by feeding H and HQ to a transformer.

Spam Prediction

Span prediction is the prediction of the start and the end probabilities of an answer. Therefore, we apply a linear layer with SoftMax activation and feed H` as input to obtain the probabilities. The training objective used for predicting the answer span to a question is cross entropy loss for start and end index prediction.

Internal Front Verification

This component of intensive reader focuses more on identifying unanswerable questions. It is implemented as cross entropy loss. Thus the pooled first token h`1 ∈ H' is fed to a fully connected layer to obtain classification logits.

Threshold based answerability verification

It is a heuristic method to decide whether a question has an answer or not according to the answer start and end logits. Let s, e, and v be start, end, and verification probabilities and we determine two scores, one is for answerable question score_has and the other is for unanswerable question score_null.
score_has = max(s_k + e_l),1 < k ≤ l ≤ n,

score_null = s1 + e1,

score_diff = score_null - score_has

A threshold δ is chosen for answerability verification. If the final score is above the threshold, answer span is predicted and score_has is given otherwise null string is given by the model. score_diff is used for the final no-answer score and later applied in rear verification.

Rear verification

The function of the rear verification module is to combine scores obtained from the sketchy reader (i.e. score_ext) and the intensive reader (i.e. score_diff) which is used for the final verification of the answer.
v = β1_score_ext + β2_score_diff
Where β1 and β2 are weights for training. If v > δ, the model predicts the answer, else it returns a null string.

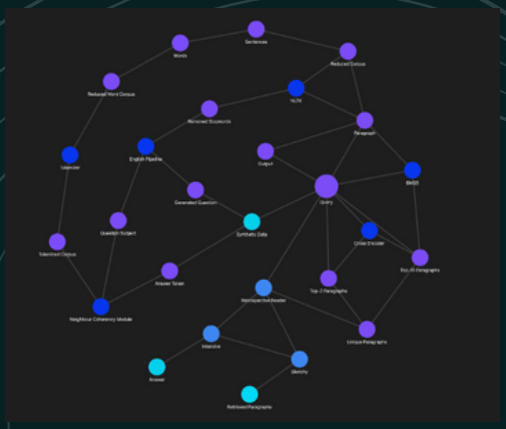
MODEL PIPELINE

Visualization of Complete Model Pipeline:

01

Grouping For Every Theme

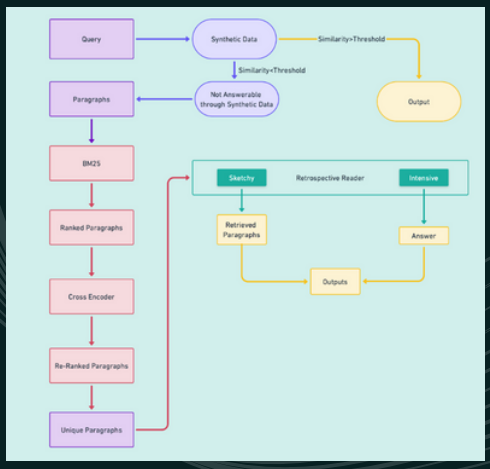
The paragraphs provided are broken down into small sentences and then finally into words. This chronological order thus gives a Single Question Per Line Approach.



02

Visual Representation of the pipeline:

Stopwords from the extracted corpus are filtered out using a NLTK corpus library.



Paragraph Retrieval

01 — Mapping With Synthetic Data

Any incoming query is mapped to the synthetic data generated. The Cosine Similarity metric was used for calculating the similarity between query and the questions present in synthetic data. If the similarity of the query and any question was found to be greater than a tunable threshold, then that particular context-question-answer triplet from the Synthetic data gets appended in a list of probable outputs. From that list, the context-question-answer triplet with the highest similarity is passed as the output. If no such instances are observed, then we move to the next stage of the pipeline.

NOTE:

Our team used the top scoring context of BM25 and the top 2 re-ranked contexts from the Cross Encoder as the input to the Retrospective Reader. The BM25 module is based on TF-IDF vectorisation but fails to take into account the contexts present in the text. On the other hand, the Cross Encoder Module focuses on the context. Thus, taking into account entries from both the modules helped us achieve a balance between TF-IDF and contexts. The retrieved paragraphs had an accuracy of 92% (containing the correct paragraph)

02 — Initial Ranking:

The input query is used to rank the set of entire documents/contexts using the BM25 algorithm. Through the set of the ranked documents, the top 10 documents (if available) with the highest probability score are taken into consideration and passed to the next stage of the pipeline, i.e., Cross Encoding.

03 — Reranking:

The set of the top retrieved documents is then passed to the Cross Encode Module, which is used to rerank the set of contexts.The Cross Encoder module then provides the top 2 most relevant contexts through which the query can be answered.

04 — Sketchy Reader:

The retrieved paragraphs along with the query are then passed to the Retrospective Reader where they first get feeded into the Sketchy Reader Module. The scoreext from the module is used to determine if the query can be answered from any of those paragraphs. If scoreext is greater than threshold, then the query is deemed answerable from the input contexts.

Answer Retrieval

01 **Intensive Reader:**
If the query is judged to be answerable from the input contexts, the query and contexts are passed into the Intensive Reader. The Intensive Reader on obtaining the interaction layer, reconfirms whether the query is answerable or not from the input contexts. If answerable, it also predicts the span.

02 **Rear Verifier:**
Rear Verifier is the final submodule of the Retrospective Reader, which compiles the answer predictability through comparison of scores obtained from the Intensive and Sketchy Modules to a tunable threshold.

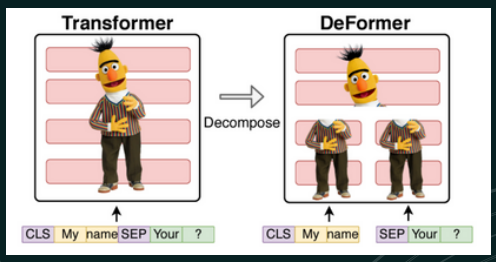
03 **Final Predictions::**
If the answer is obtained through the Retrospective Reader, then the index of the paragraph from which the span was taken is passed as the output of paragraph retrieval and the answer span as the answer of the query. If not, then the paragraph index -1 is passed with an empty string as an answer.

Optimizations Made

01

Employment of Deformer architecture:

The DeFormer model is a variant of Transformer-based models that utilizes self-attention in a more efficient manner by decomposing the attention process to apply it on the input question and passage individually. This allows for more independent text interpretation at lower layers and reduces runtime significantly with a small cost in accuracy scores. The deformer architecture thus can be implemented naively by effectively changing last layers of a model to work with a different architecture. Thus a Roberta Model working on an Electra Architecture serves for a simple deformer layout.



02

Transformer Compression:

We used transformer compression methods in our model for optimizing the pipeline. These two methods are:

1. Quantization
2. Prunification

We have quantized our model to change the model weights type from float64 to int8. (See Appendix)

Domain Adaptor

To adapt our model for any new domain, we have employed three main approaches:

01

Using already answered question answer pairs to map any new query by checking similarity scores of the query and previously answered questions. This forms a feedback loop that enters question answer pairs in the context-question-answer database.

02

Generating a synthetic dataset for every theme. Increasing the amount of questions generated, increases the chances of any incoming question to have been already answered. Thus, this increases domain specificity.

03

Training of the model can be achieved in two possible ways:

1. Training the model for the entire set of themes
2. Training various models for different themes

Our team implemented both approaches and found general training over the entire theme set to be the most optimal due to the following reasons:

A, Less Memory Consumption

B. No Latency and no adverse effects on RAM while loading general model for all themes

C. Easier Training (See Training Results)

Results and Runtime Analysis

Training

Fine Tuning

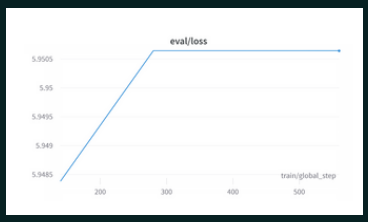
Fine Tuning can distort pretrained weights and the fine tuned model can underperform than the pretrained general model if the training data is not sufficient and the training time given to finetune is not sufficient. As per the P.S. constraints the time provided to us for fine tuning and generating synthetic data for 30 themes was 6 hours which gave us around 10 to 15 min for fine tuning.

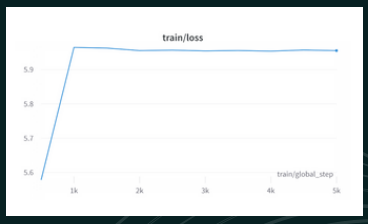
F1 score given by a general pretrained model on a single theme: 0.9221263886515272

F1 score given by a fine tuned model(fine tuned on that theme) : 0.8920588462512887

Time taken for fine tuning : 21 min 32 sec

Epochs = 3 (running for more epochs would have crossed the time constraints)





Syn data gen:

Generated 1379 Questions for 239 Paragraphs.

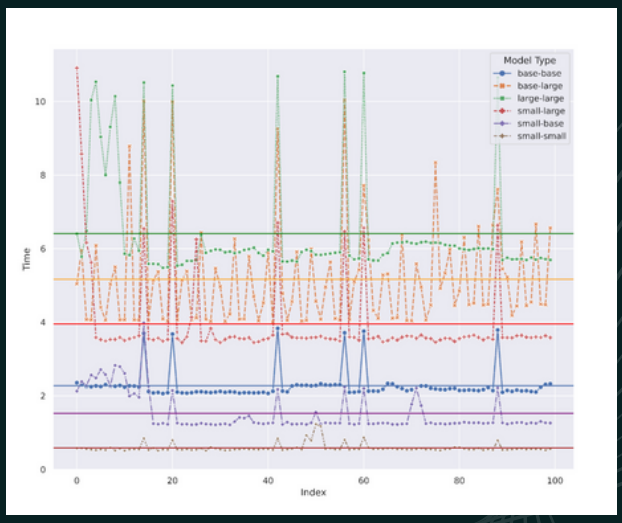
Time Taken for Generation = 45 min 58 sec

F1 score of the generated question answer pairs = 0.8085547850707818

(This F1 score was generated by comparing generated answers with answers given by large Question/Answering Models.)

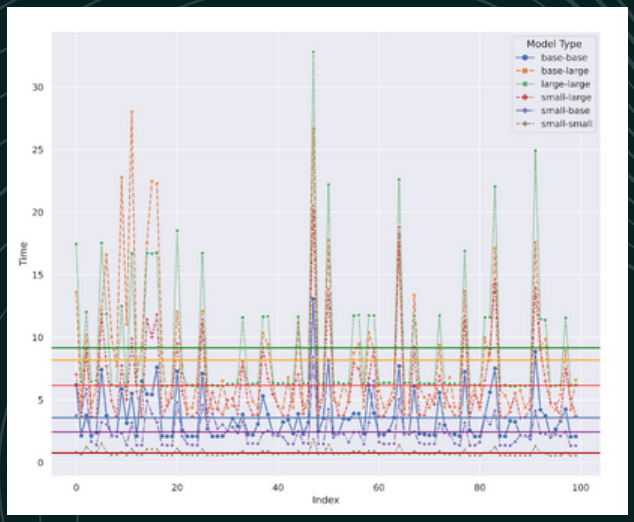
Inference

Inference time of retrospective readers for sketchy and intensive as (base, base), (base, large), (large, large), (small, large), (small, base), (small, small) respectively. One can notice a continuous decrease in inference times when the electra models are reduced from large to base to small. Intensive reader contributes to a larger change when intensive reader models are changed.

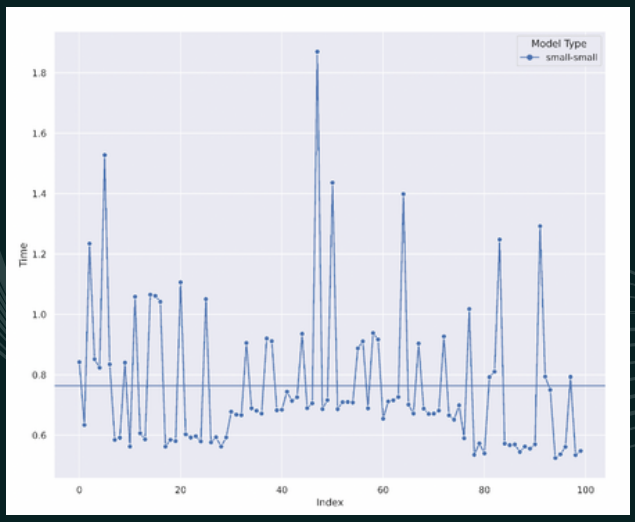


SKETCHY READER MODEL	INTENSIVE READER MODEL	F1 SCORE	PARAGRAPH ACCURACY (EM)	AVERAGE INFERENCE TIME (MS)
Electra-small	**Electra-small**	**0.86**	**0.89**	**764 ms**
Electra-small	Electra-base	0.91	0.90	2422 ms
Electra-small	Electra-large	0.88	0.89	6148 ms
Electra-base	Electra-base	0.91	0.90	3576 ms
Electra-base	Electra-large	0.88	0.89	8176 ms
Electra-large	Electra-large	0.88	0.89	9154 ms

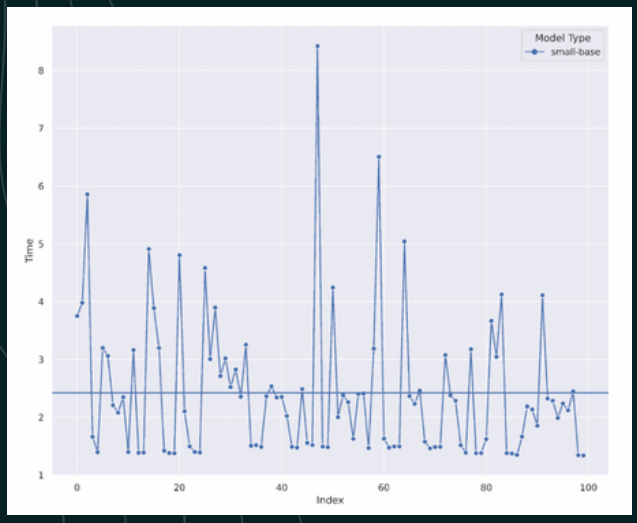
Inference time of retrospective reader+bm25+ce+ques matching using synthetic data.



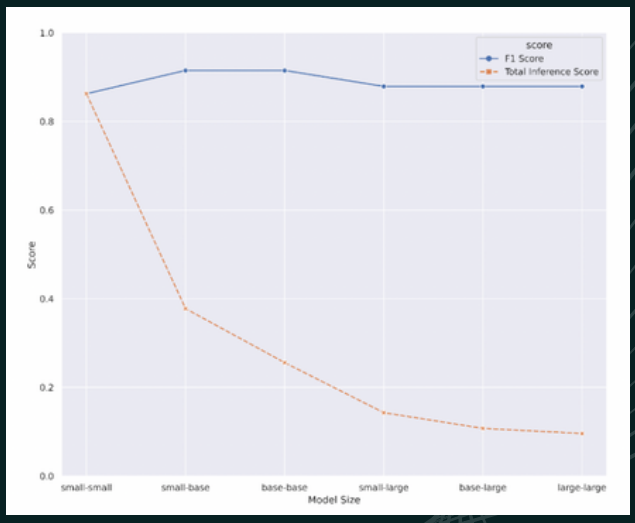
Inference time of retrospective reader+bm25+ce+ques matching using synthetic data (small-small).



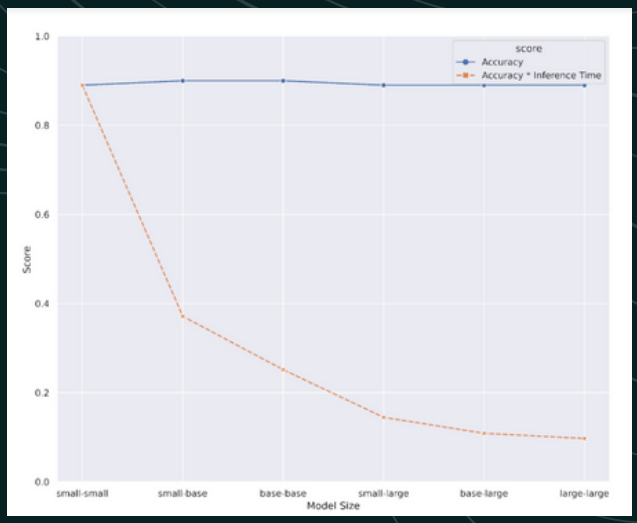
Inference time of retrospective reader+bm25+ce+ques matching using synthetic data (small-base).



F1 score of retrospective reader+bm25+ce+ques matching using synthetic data.



Para retrieval accuracy of retrospective reader+bm25+ce+ques matching using synthetic data.



CONCLUSION

During our entire work on building an efficient and optimal question answering system, we tried and tested different implementations with the basic designs varying from simple algorithms for contextual retrieval to more complex architectures consisting of multiple units, as seen in the implementation of Retrospective Reader. We also tested several paragraph retrieval algorithms and finally came up with an optimal solution to it.Our focus was to make a state-of-the-art model with high F1 scores on a variety of domains with a relatively lesser inference time. We successfully achieved this target as our current model efficiently adapts to different domains and yields a significantly lower inference time (approximately 1 second). We can confidently state that our model can be easily deployed with lesser computational costs in real-world applications.

APPENDIX

1. ONXX Runtime

Open Neural Network Exchange (ONNX) ecosystem provides for models that are aimed at optimizing artificial neural networks. ONNX models also give the users the freedom to write code without having to specifically design it for different hardware since such models are independent of CPU/GPU usages. Having no dependency on hardware, such models enable the deployment much easier..ONNX models use different techniques that enable them to optimize their performance, such as:

1. Just in Time compilation
2. Kernel Fusion
3. Subparagraph Partitioning

Using these ONNX based models also takes significantly less time training for the same number of epochs and batch size.Our team has also been working on converting our codebase to use ONNX models. These models are expected to decrease our runtime without compromising performance. The only issue that came up while deploying such ONNX models in our implementation was the unavailability of pretrained models that could be easily used within the deadline of our project.

2. Knowledge Graph

Knowledge Graphs are search systems that are used for enhancing semantic-search information gathered from a wide variety of sources. Information about different entities is linked into a graph based structure and thus search queries form links with these entities to develop a relationship.

The methodology on knowledge graphs was implemented for paragraph retrieval. The following technical structure was used:

1. Preprocessing: Tokenize the questions and paragraphs into words and perform necessary preprocessing steps such as removing stop words, stemming, etc.

2. Graph creation: Create a graph using NetworkX (a graph creation and visualization library) where each node represents a word in the question or paragraph and edges are created based on the similarity between words.Later weights were assigned to each edges to give more importance to certain subject and answer pair tokens

3. Similarity calculation: Calculate the similarity between words using a similarity metric such as Cosine similarity. The similarity scores can be used to create edges between words in the graph.

4. Question-paragraph mapping: Use a graph-based algorithm such as betweenness centrality to find the most relevant paragraph to the question. Betweenness centrality is a measure of centrality in a graph that calculates the number of times a node acts as a bridge along the shortest path between two other nodes. It measures the node's influence on the flow of information or resources in the network by quantifying the proportion of all shortest paths that pass through that node.

The calculation of betweenness centrality for a node follows this procedure:
1. Using Dijkstra's algorithm to find the shortest path between each pair of nodes
2. Count the number of shortest paths common between this node and all other nodes.
3. Normalizing the above count by dividing by the total number of node pair edges.

The betweenness centrality score of a node represents the proportion of shortest paths that pass through that node. Nodes with high betweenness centrality scores are more likely to play an important role in the flow of information or resources in the network and are considered to be more central.

It is worth noting that the betweenness centrality calculation can be computationally expensive, especially for large graphs, as it requires finding the shortest paths between all pairs of nodes.

5. In the context of mapping a question to a paragraph from a set of paragraphs using NetworkX, betweenness centrality can be used as an algorithm to calculate the importance of each word in the graph. The words that have high betweenness centrality scores are more likely to be relevant to the question and therefore can be used to determine the most relevant paragraph.The algorithm scores each paragraph based on the connectivity of its words to the words in the question.

6. Paragraph selection: We select the paragraph with the highest score as the answer to the question.

```python
import networkx as nx
import spacy
from sklearn.cluster import SpectralClustering
import numpy as np

nlp = spacy.load("en_core_web_sm")

def extract_entities(text):
    doc = nlp(text)
    entities = [entity.text for entity in doc.ents]
    return entities

def extract_relationships(text):
    doc = nlp(text)
    relationships = [(token.text, token.head.text) for token in doc if token.dep_ in ["nsubj", "dobj"]]
    return relationships

def build_knowledge_graph(paragraphs):
    graph = nx.Graph()
    for paragraph in paragraphs:
        entities = extract_entities(paragraph)
        relationships = extract_relationships(paragraph)
        for entity in entities:
            graph.add_node(entity)
        for relationship in relationships:
            graph.add_edge(relationship[0], relationship[1])

    adjacency_matrix = nx.to_numpy_array(graph)
    adjacency_matrix = adjacency_matrix + 1e-6 * np.eye(adjacency_matrix.shape[0])
    adjacency_matrix = adjacency_matrix / np.linalg.norm(adjacency_matrix, ord=2, axis=1, keepdims=True)
    clustering = SpectralClustering(n_clusters=2, affinity='precomputed', assign_labels='kmeans')
    clustering.fit(adjacency_matrix)

    for i, entity in enumerate(graph.nodes):
        graph.nodes[entity]['cluster'] = clustering.labels_[i]
    return graph

def find_relevant_paragraph(question, paragraphs, graph):
    question_entities = set(extract_entities(question))
    question_relationships = set(extract_relationships(question))
    scores = [0] * len(paragraphs)
    entity_lookup = {}
    for i, paragraph in enumerate(paragraphs):
        paragraph_entities = set(extract_entities(paragraph))
        paragraph_relationships = set(extract_relationships(paragraph))
        scores[i] = len(question_entities.intersection(paragraph_entities))+len(question_relationships.intersection(paragraph_relationships))
        for entity in question_entities:
            if entity in entity_lookup:
                scores[i] += len(entity_lookup[entity].intersection(paragraph_relationships))
            else:
                entity_relationships = set()
                for relationship in graph.edges(entity):
                    entity_relationships.add(relationship)
                entity_lookup[entity] = entity_relationships
    return paragraphs[scores.index(max(scores))]

def map_question_to_paragraph(question, paragraphs, graph):
    relevant_paragraph = find_relevant_paragraph(question, paragraphs, graph)
    return relevant_paragraph
```

3. Deformers

If there are n layers in a model and we decide to run self attention separately till k layers. Before runtime, the text is run through the k layers and the output is cached thus reducing the memory usage. During runtime, after the question has been processed through the k layers, the cached output is loaded and the processing continues similar to the original transformer from the k+1th layer.
A = [a1; a2;...; aq] Embeddings for first segment
B = [b1;b2;...;bp] Embeddings for second segment
X = [A; B] Input Sequence
XI+1 = Li(XI) Input where Li denotes the ith layer
[An;Bn] = Lk+1:n([L1:k(A0); L1:k(B0)]) Output representation of full transformer where Lk+1:n denotes application of k+1 to n layers
The time complexity for each lower layer is reduced from O((p+q)2) to O(q2+c) where p and q are passage and question respectively and c represents a constant
Knowledge Distillation Loss: It involves minimizing Kullback—Leibler divergence between decomposed Transformer prediction distribution PA and full Transformer prediction distribution PB.
Lkd = DKL(PA||PB)
Layerwise Representation Similarity Loss: It involves minimizing the euclidean distance between token representations of the upper layer of the original transformer and the decomposed transformer.

4. Simple TF-IDF vectorization and dictionary mapping of Nouns,Adverbs, Adjectives, Verbs for paragraph rerieval

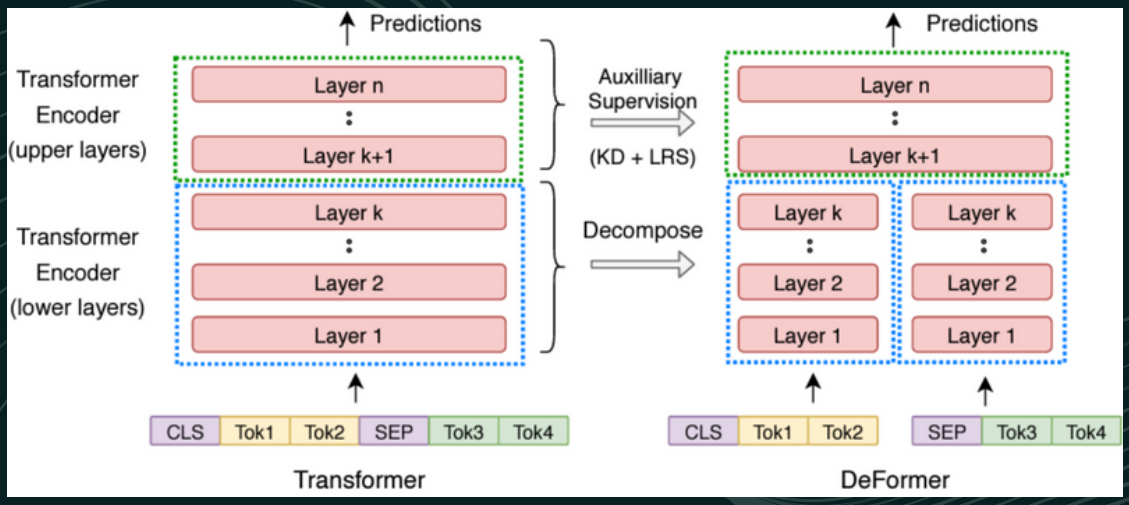
The method maps a query to the most relevant paragraph from a series of paragraphs by combining NLP methods with information retrieval algorithms.

The following is the algorithm:
1. NLTK libraries are loaded to accomplish tasks like tokenization and POS tagging.
2. The nltk Wordnet corpus is utilised for lemmatization.
3. The sklearn library's cosine similarity and TfidfVectorizer are used to calculate cosine similarity and generate term frequency-inverse document frequency (TF-IDF) values, respectively.
4. To identify the POS tag of each word in the text, a function is utilised. This data is then utilised to calculate the lemma for each word.
5. A lemmatization function takes a string of text as input and returns the lemmatized version of the text. The above uses the WordNetLemmatizer from nltk to perform lemmatization.
6. The lemmatized versions of the paragraphs and the question are stored in the paragraphs and question variables respectively.
7. We convert the paragraphs and the question to a matrix of TF-IDF values and a TfidfVectorizer object is created.
8. The resulting matrix is stored in the vectors variable and then we calculate the cosine similarity between the question vector and each paragraph vector.
9. The highest similarity mapping is used to store as the most relevant paragraph.

$$\mathcal{L}_{lrs} = \sum_{i=k}^{n} \sum_{j=1}^{m} \|\mathbf{v}_j^i - \mathbf{u}_j^i\|^2$$

Ltotal = γLts + αLkd +βLlrs
Ltotal = total loss
Lts = task specific supervision loss
Lkd = knowledge distillation loss
Llrs = layerwise representation similarity loss
γ, α and β are obtained by hyper-parameter tuning using Bayesian Optimization.

```python
import nltk
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer

nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('omw-1.4')

# Define the set of paragraphs
paragraphs = [
    "This is the first paragraph. It contains some information about apples.",
    "This is the second paragraph. It talks about the benefits of eating apples.",
    "This is the third paragraph. It describes how to make apple pie."
]

# Define the question
question = "What are the benefits of eating apples?"

# Define a function to perform lemmatization
def get_wordnet_pos(word):
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

def lemmatize_text(text):
    lemmatizer = WordNetLemmatizer()
    return " ".join([lemmatizer.lemmatize(w, get_wordnet_pos(w)) for w in nltk.word_tokenize(text)])

# Perform lemmatization on paragraphs and question
paragraphs = [lemmatize_text(p) for p in paragraphs]
question = lemmatize_text(question)

print(paragraphs)
print(question)
# Convert paragraphs and question to a matrix of TF-IDF values
vectorizer = TfidfVectorizer()
vectors = vectorizer.fit_transform(paragraphs + [question])

# Calculate cosine similarity between question vector and each paragraph vector
question_vector = vectors[-1]
similarities = cosine_similarity(question_vector, vectors[:-1])

print(question_vector.shape,vectors[:-1].shape)
# Find the index of the most similar paragraph
most_similar_index = similarities.argmax()

# Print the most similar paragraph
print(paragraphs[most_similar_index])
```



5. BM25 Ranking Algorithm

BM25 is a bag of words retrieval function that ranks a set of documents based on their TF-IDF. BM25 calculates the score of a document D as:

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i,D)\cdot(k_1+1)}{f(q_i,D)+k_1\cdot\left(1-b+b\cdot\frac{|D|}{\text{avgdl}}\right)}$$

D: document/paragraph
Q: Query
qi: word in the query
k_1 , b : Hyperparameters
Avgdl: average document length from text collection
IDF : Inverse Document Frequency
|D| : Document Length

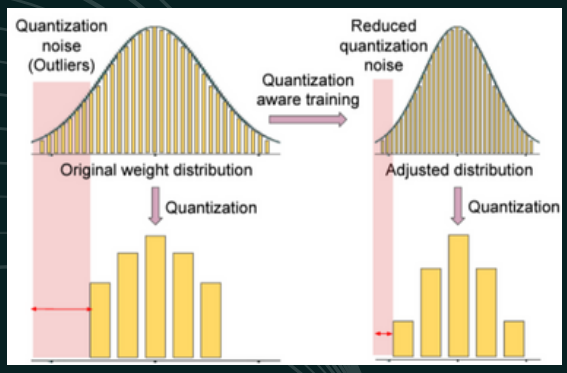
6. Cross Entropy

$$L^{ans} = -\frac{1}{N}\sum_{i=1}^{n}\left[\, y_i \log \hat{y}_i + \left(1-y_i\right)\log\left(1-\hat{y}_i\right)\right]$$

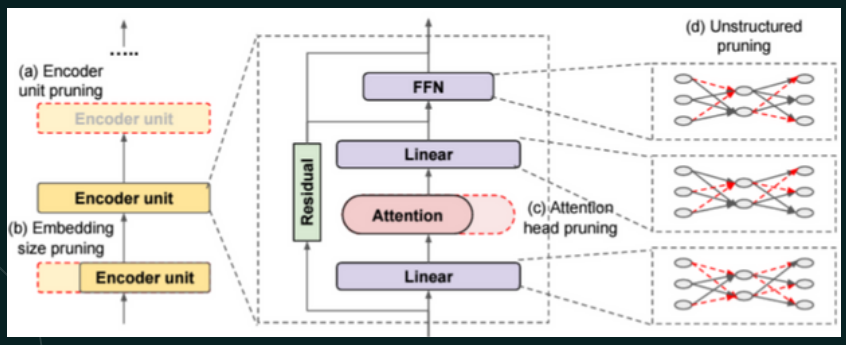
Where yi is the actual value showing whether the question is answerable or not and yi∝ SoftMax(FFN (h1)) is the prediction. N is the number of instances.
scoreext = logitna - logitans denotes the external front verification score which will be later used in the rear verification module.

7. Transformer Compression

The transformers have billions of parameters which limit its usage on low-capability devices. Transformer compression overcomes this problem by using a fraction of the computational resources, i.e., shrinks the trained neural network. Following are the two transformer compression methodologies we have explored:
Quantization: Reduces the number of unique values representing the weights so as to represent them in fewer bits which makes the model both memory efficient and time efficient if the computational device is optimized to process lower-precision numerical values.



- Pruning: Involves identifying and removing redundant and less important weights which can even improve the performance of the model. There are two types of pruning:
  1. Unstructured Pruning: Also known as sparse pruning, unstructured pruning prunes individual weights using their absolute values, gradients, or some custom measurement method, removing the least important ones.
  2. Structured Pruning: Involves pruning entire structured blocks of weights like attention head pruning, encoder unit pruning and embedding size pruning, removing attention heads, encoder layers and feature dimensions respectively.



We have quantized our model to change the model weights type from float64 to int8.

References

[1] Zhang Z, Yang J, Zhao H. Retrospective reader for machine reading comprehension. InProceedings of the AAAI Conference on Artificial Intelligence 2021 May 18 2001.09694v4.pdf (arxiv.org)

[2] Stephen Robertson and Hugo Zaragoza (2009), "The Probabilistic Relevance Framework: BM25 and Beyond", Foundations and Trends® in Information Retrieval: Vol. 3: No. 4, pp 333-389. http://dx.doi.org/10.1561/1500000019

[3] Lopez, L.E., Cruz, D.K., Cruz, J.C.B., Cheng, C. (2021). Simplifying Paragraph-Level Question Generation via Transformer Language Models. In: Pham, D.N., Theeramunkong, T., Governatori, G., Liu, F. (eds) PRICAI 2021: Trends in Artificial Intelligence. PRICAI 2021. Lecture Notes in Computer Science(), vol 13032. Springer, Cham. https://doi.org/10.1007/978-3-030-89363-7_25

[4] Aithal, S.G., Rao, A.B. & Singh, S. Automatic question-answer pairs generation and question similarity mechanism in question answering system. *Appl Intell* 51, 8484–8497 (2021). https://doi.org/10.1007/s10489-021-02348-9

[5] Qi W, Yan Y, Gong Y, Liu D, Duan N, Chen J, Zhang R, Zhou M (2020) Prophetnet: Predicting Future N-gram for Sequence-to-Sequence Pre-training In: Findings of the association for computational linguistics: EMNLP 2020. Association for Computational Linguistics, Online, pp 2401–2410. https://doi.org/10.18653/v1/2020.findings-emnlp.217

[6] Devlin J, Chang MW, Lee K, Toutanova K (2019) BERT: Pre-training Of Deep Bidirectional Transformers for Language Understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). Association for Computational Linguistics, Minneapolis, pp 4171–4186.
https://doi.org/10.18653/v1/N19-1423

[7] Bollacker K, Evans C, Paritosh P, Sturge T, Taylor J (2008) Freebase: A collaboratively created graph database for structuring human knowledge. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD '08. Association for Computing Machinery, New York, pp 1247–1250.
https://doi.org/10.1145/1376616.1376746

[8] Wendy G. Lehnert. 1977. A conceptual theory of question answering. In Proceedings of the 5th international joint conference on Artificial intelligence - Volume 1 (IJCAI'77). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 158–164.
ERIC - ED150955 - The Process of Question Answering. Research Report No. 88., 1977-May

[9] Sukhbaatar S, Weston J, Fergus R. End-to-end memory networks. Advances in neural information processing systems. 2015;28.
1503.08895.pdf (arxiv.org)

[10] Ho, M., Zhou, Z. and He, J., 2021. When to Fold'em: How to answer Unanswerable questions. *arXiv preprint arXiv:2105.00328*.
2105.00328.pdf (arxiv.org)

[11] Levy O, Seo M, Choi E, Zettlemoyer L. Zero-shot relation extraction via reading comprehension. arXiv preprint arXiv:1706.04115. 2017 Jun 13.
1706.04115.pdf (arxiv.org)

[12] Cao Q, Trivedi H, Balasubramanian A, Balasubramanian N. DeFormer: Decomposing pre-trained transformers for faster question answering. arXiv preprint arXiv:2005.00697. 2020 May 2.
[2005.00697] DeFormer: Decomposing Pre-trained Transformers for Faster Question Answering (arxiv.org)

[13] Gupta M, Varma V, Damani S, Narahari KN. Compression of deep learning models for NLP. InProceedings of the 29th ACM International Conference on Information & Knowledge Management 2020 Oct 19 (pp. 3507-3508).
Compression of Deep Learning Models for NLP | Proceedings of the 29th ACM International Conference on Information & Knowledge Management

[14] Cao Y, Fang M, Yu B, Zhou JT. Unsupervised domain adaptation on reading comprehension. InProceedings of the AAAI Conference on Artificial Intelligence 2020 Apr 3 (Vol. 34, No. 05, pp. 7480-7487).
[1911.06137] Unsupervised Domain Adaptation on Reading Comprehension (arxiv.org)

[15] H. -G. Lee, Y. Jang and H. Kim, "Machine Reading Comprehension Framework Based on Self-Training for Domain Adaptation," in IEEE Access, vol. 9, pp. 21279-21285, 2021, doi: 10.1109/ACCESS.2021.3054912.
Machine Reading Comprehension Framework Based on Self-Training for Domain Adaptation | IEEE Journals & Magazine | IEEE Xplore

[16] Pradeep R, Liu Y, Zhang X, Li Y, Yates A, Lin J. Squeezing Water from a Stone: A Bag of Tricks for Further Improving Cross-Encoder Effectiveness for Reranking. InAdvances in Information Retrieval: 44th European Conference on IR Research, ECIR 2022, Stavanger, Norway, April 10–14, 2022, Proceedings, Part I 2022 Apr 5 (pp. 655-670). Cham: Springer International Publishing.
Squeezing Water from a Stone: A Bag of Tricks for Further Improving Cross-Encoder Effectiveness for Reranking | SpringerLink

[17] Izacard G, Caron M, Hosseini L, Riedel S, Bojanowski P, Joulin A, Grave E. Unsupervised dense information retrieval with contrastive learning.
pdf (openreview.net)

[18] Lin J. A proposed conceptual framework for a representational approach to information retrieval. InACM SIGIR Forum 2022 Mar 17 (Vol. 55, No. 2, pp. 1-29). New York, NY, USA: ACM.
A proposed conceptual framework for a representational approach to information retrieval | ACM SIGIR Forum