DROOID ASSIGNMENT

# ML - Question

NAME: MOHIT SHARMA

---

## 1. <u>Preprocessing:</u>

- Functions `preprocess` and `extract_keywords` clean and prepare the text data for further processing. This includes removing special characters, converting to lowercase, and removing stop words.

```python
def extract_keywords(question):
    stop_words = set(stopwords.words('english'))
    words = word_tokenize(question)
    keywords = [word.lower() for word in words if word.isalnum() and word.lower() not in stop_words]
    return keywords
```

## 2. <u>Article Retrieval:</u>

- `get_relevant_article` takes a question as input.

```python
def get_relevant_article(question):
    print("getting relevent document")
#     # Clean and tokenize the articles
    keywords = extract_keywords(question)

    filtered_articles = [article for article in articles if 'Israel' in article['title'] or 'Hamas' in article['title']]
    tokenized_articles = [preprocess(article['articleBody']) for article in filtered_articles if any(keyword in article['articleBody'].lower() for keywo

    # Initialize BM25
    bm25 = BM25Okapi(tokenized_articles)

    # Retrieve top 5 relevant articles
    top_articles = retrieve_documents(question, bm25, articles, n=5)

    # Print the titles of the top articles
    for i, article in enumerate(top_articles):
        print(f"Article {i+1}: {article['title']}")

    # Choose the most relevant article (first one)
    relevant_article = top_articles[0]
    print("relevant_article returned - Article Length")
    return relevant_article
```

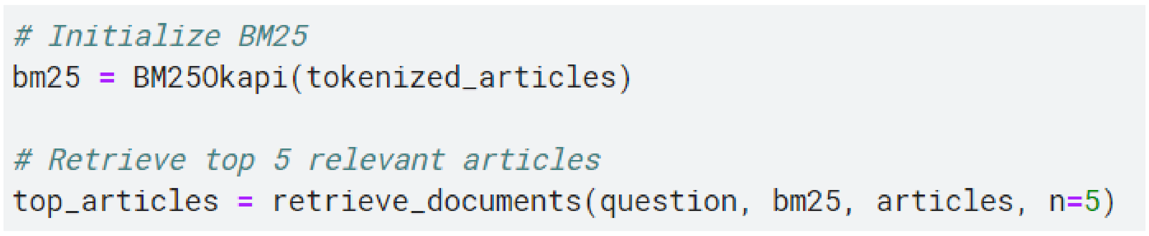
- It first extracts keywords from the question and filters articles containing "Israel" or "Hamas" in the title to improve efficiency.

```python
filtered_articles = [article for article in articles if 'Israel' in article['title'] or 'Hamas' in article['title']]
```

- Then, it cleans and tokenizes the remaining articles' bodies.

```python
tokenized_articles = [preprocess(article['articleBody']) for article in filtered_articles \
                      if any(keyword in article['articleBody'].lower() for keyword in keywords)]
```

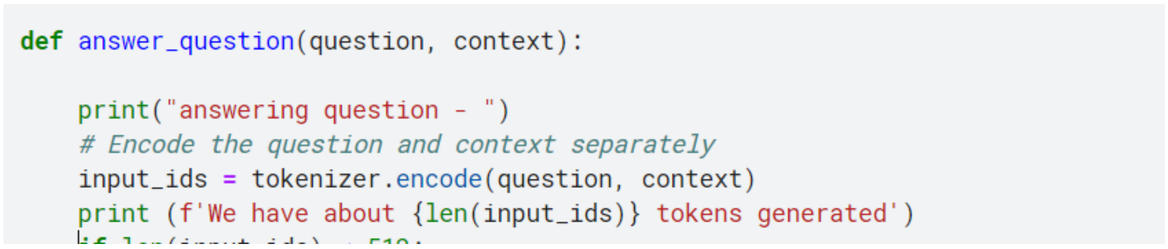
- A BM25 ranking model is used to score these articles based on their relevance to the question.
- Finally, the top 5 articles are retrieved.



```python
# Initialize BM25
bm25 = BM25Okapi(tokenized_articles)

# Retrieve top 5 relevant articles
top_articles = retrieve_documents(question, bm25, articles, n=5)
```

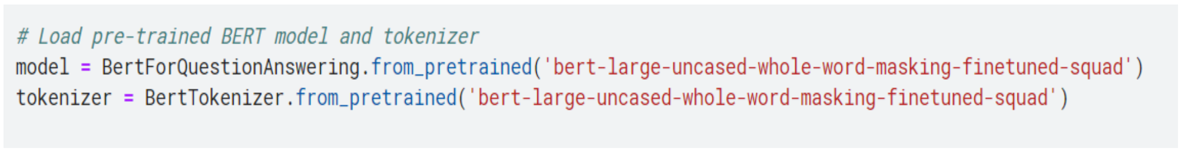
## 3. Question Answering:

- `answer_question` takes a question and the relevant article's body as input.



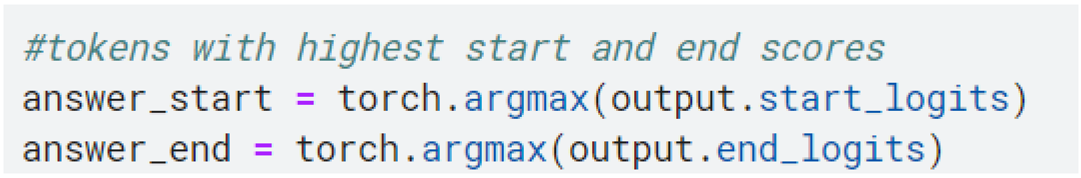
```python
def answer_question(question, context):

    print("answering question - ")
    # Encode the question and context separately
    input_ids = tokenizer.encode(question, context)
    print (f'We have about {len(input_ids)} tokens generated')
```

- It uses a pre-trained BERT model to process the question and context together.



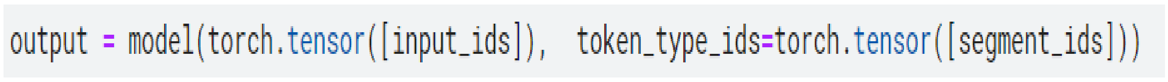
```python
# Load pre-trained BERT model and tokenizer
model = BertForQuestionAnswering.from_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')
tokenizer = BertTokenizer.from_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')
```

- The model predicts the most likely answer span (start and end positions) within the context for the question.

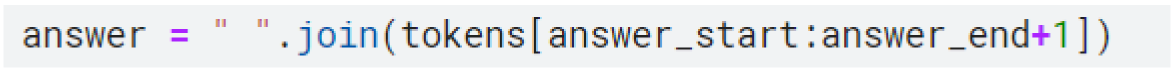


```python
#tokens with highest start and end scores
answer_start = torch.argmax(output.start_logits)
answer_end = torch.argmax(output.end_logits)
```

- Finally, the answer text is extracted from the article's body based on the predicted span.



```python
output = model(torch.tensor([input_ids]),  token_type_ids=torch.tensor([segment_ids]))
```



```python
answer = " ".join(tokens[answer_start:answer_end+1])
```

**Overall, the code combines techniques for information retrieval and machine reading comprehension to answer questions based on a document collection.**