Statistical Learning: Project – Question Answering

Benjamin Cohen, Ilyass Belkhayat, Guillaume Nodet

July 6, 2024

Abstract

This project is about answering questions using language models that have already been trained. We studied several different models, ChatGPT, Llama2, GPT2, Pure Information Retrieval Exact-Match using DuckDuckGoSearchRun with DeBERTaV3 for answer extraction and BM25 using Pyserini to see how well they can understand and answer questions using natural language, find information from knowledge bases, and give good responses. The NQ-Open dataset will be used containing questions and their answers from Wikipedia.

We are looking at different ways to solve the problem. The analysis uses extractive question answering models, the BM25 model, and looks at how neural models are indexed. Models like GPT2 are integrated for particular purposes.

Performance is checked by comparing to a set of correct answers. We compare the different models and discuss where they perform and where they do not. The report talks about the different models, what happened in the experiments, what issues came up, and how things could be better next time.

The project's practical side shows in trying out and checking different methods. The project also shows the moral issues in using those models.

1 Introduction

In the area of AI, pre-trained language models have changed the way we work on understanding and using natural language. Out of these models, ChatGPT, Llama2, and GPT2, which are all made by OpenAI, are good at understanding and creating text. This project will study these models in detail, especially looking at the difficult job of answering questions.

Answering questions is an important part of understanding language. To do this, a computer needs to be able to understand the question, look for the answer in a database, and give the correct answer. We are using the NQ-Open dataset for our research. It has lots of examples of questions and their answers from Wikipedia.

We also use different ways to do this AQ task, using many kinds of models and techniques. Our project wants to understand how different computer models answer questions in different ways.

There are different ways to do this, like using Pure information-retrieval, Probabilistic model: BM25 like BM25, Neural models and Tool-based answering.

2 Literature Review

Pre-trained language models are language models that have been trained before and have changed the way we work with language by understanding and representing its meaning in a very detailed way. The early work of models like BERT [5] and GPT [1] has raised the bar for how well computers can understand and process language in different tasks, including answering questions.

The GPT2 model [2], a type of Transformer, has been used a lot for AQ. It's good at making text even in hard situations, so lots of people like it. Research done by Brown and others. In 2020, GPT2 showed how well it can answer questions, proving that it can be used for many different problems.

ChatGPT [4] is designed for talking, chatting with people and to have conversations. Research like the one conducted by Radford and his team. In 2019, it was found that ChatGPT is good at coming up with sensible answers in ongoing conversations. Yet, it hasn't been used much for specific AQ tasks.

Llama2 [6] is different because it has a unique way of finding information. It was created to find useful information from databases, Llama2 concentrates on finding information.

New improvements have been made in using pre-trained language models for AQ, but there are still difficulties to overcome. Researchers are still studying how to handle different situations, give the same answers all the time, and understand difficult questions.

3 Dataset

The NQ-Open dataset is a dataset designed for the Question Answering (QA) task and is available at the following address: NQ-Open on Hugging Face. The dataset is divided into two main subsets: "training" and "validation". The training set contains 87,925 elements and the validation set 3610 elements. Each example in the dataset is structured as a question-answer pair. All questions have the particularity of having their answers available on Wikipedia. The dataset includes a variety of questions covering different areas of knowledge and linguistic complexity. Answers are structured as lists, reflecting the possibility of multiple correct answers. Performance is evaluated using the "Exact Match" method. An answer is considered correct only if it matches exactly (to the nearest square) the expected answer. The NQ-Open dataset contains a diversity of questions and answers. Its use in this project offers an opportunity to explore and implement effective approaches to solving the complex task of Question Answering.

4 Models description

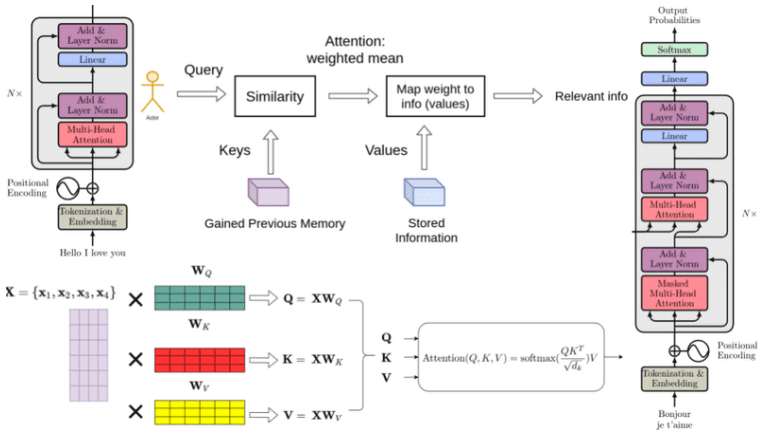

Figure 1: Transformer Architecture

The Transformer architecture, introduced by Vaswani et al. in Attention Is All You Need (2017) [3], is fundamental in the field of natural language processing (NLP). It has revolutionized sequential modeling by eliminating the need to use recurrent or convolutional architectures, offering a more parallel and efficient alternative. The Transformer architecture was developed to solve the limitations of recurrent network-based models in the processing of long sequences.

4.1 Main components of the Transformer architecture

4.1.1 Multi-headed attention

Enables the model to dwell on different parts of the sequence simultaneously by dividing the embedding space into several heads, each calculating independent attention. In traditional attention mechanisms, a single set of learnable parameters is used to compute the attention scores between the input elements (usually tokens in a sequence). The attention mechanism computes attention weights for each input element based on its relevance to a query.

We use:

- **Query (Q)**: The element for which we want to compute attention scores.
- **Key (K)**: The elements used to determine the importance of the query.

- **Value (V)**: The values associated with the keys, which are weighted by attention scores.

The idea behind multi-headed attention is to use multiple sets of learnable parameters (heads) for computing attention scores independently. Each head performs a linear transformation on the input queries, keys, and values. The outputs of the different heads are concatenated and linearly transformed again to produce the final output.

Given an input sequence of dimension d_{model} (model dimension), the multi-headed attention is computed as follows:
MultiHead(Q,K,V)=Concat(head$_1$,...,head$_h$) * W$_0$ with head$_i$ = Attention(QW$_{Q_i}$, KW$_{K_i}$, VW$_{V_i}$)
Where QW$_{Q_i}$, KW$_{K_i}$, VW$_{V_i}$ are learnable weight matrices for the query, key, and value transformations in the i-th head. Different heads can focus on different aspects of the input, capturing diverse patterns and relationships and it allows the model to learn more complex and nuanced representations. In summary, multi-headed attention in the Transformer architecture provides a mechanism for capturing diverse relationships in the input sequence, leading to more expressive and powerful representations. It has become a fundamental building block in various natural language processing tasks and has contributed to the success of Transformer-based models.

4.1.2 Encoder-Decoder with Attention and Positional Encoding

Two distinct parts, the encoder and the decoder, with specific attention mechanisms. The encoder receives input and generates a contextual representation for each position. The decoder uses the encoding of the input to generate the output, focusing on the relevant parts. Transformers do not include information on the position of words in a sequence. It solves this problem by introducing positional embedding to inform the model of the relative position of words.

4.1.3 Attention

The attention mechanism is a way for a machine to focus on certain parts of the input data when making decisions. The attention score is computed by multiplying the query, key, and value embedding together. Scores are then divided by a certain number to make them fair. This helps to adjust for differences in the size of the space being used. We add feed-forward networks which are neural networks where data travels only from input to output, without any feedback loops. It adds complexity to the model by using linear changes and then applying specific functions for each value which are placed after each attention layer. Using layer normalization makes training more steady and reliable. We train the model with back-propagation and adjust the weights of the model. To optimize the parameters we usually use stochastic gradient descent (SGD) or Adam. The loss function is usually used for certain jobs like predicting language (next word or masked words) or translating.

4.2 Applications, Benefits and Restrictions

Natural Language Processing (NLP) is used for things like translating languages, creating text, and answering questions. There are many benefits of the transformer architecture like parallelism which allows us to do many calculations at the same time, making training faster. It is also great for long-distance relationship capture by capturing faraway connections in sequences. But there are also downsides like understanding where something is located. Indeed even though position encoding helps us understand sequences better, it still has trouble understanding the order of things. The Transformer design has been really important in improving models for understanding language. It's a strong and fast alternative to older ways of processing language. The impact can be seen in the way many language models like GPT and BERT are designed.

4.3 ChatGPT

ChatGPT is a special kind of chat program that is designed for having conversations. It is made using GPT 3.5. It has a lot of complex parts and is one of the most advanced language models. It uses the Transformer architecture and is adapted to conversations, giving natural responses that fit the situation. It possesses about 175 billion parameters and is initially trained on a wide range of text to understand a variety of languages. We can use it easily in many different conversations with OpenAI API. Some restrictions are that it might struggle with longer conversations over time. In

this subsection we will focus on the most recent language model developed by OpenAI that is freely available : ChatGPT 3.5. Since it is not possible to use it locally on a computer, we will have to use an API to communicate with OpenAI's servers and then perform our work.

Here is a quick overview of ChatGPT's features :

- **Transformer Architecture**: GPT-3.5, like its predecessors, is based on the Transformer architecture. The Transformer architecture is known for its effectiveness in handling sequential data, making it well-suited for natural language understanding and generation tasks.
- **Pre-training**: GPT-3.5 is pre-trained on a massive corpus of text data from the internet. During pre-training, it learns to predict the next word in a sentence, which helps it acquire a deep understanding of language and context. This pre-training is crucial for its performance in various natural language processing tasks.
- **Generative Model**: GPT-3.5 is a generative model, which means it can generate human-like text based on the input it receives. It can be used for a wide range of text generation tasks, including chat bots, content generation, translation, summarizing, and more.
- **Large Scale**: GPT-3.5 is characterized by its enormous scale. It has a vast number of parameters, which enables it to capture intricate patterns in language. The size of the model allows it to generate coherent and contextually relevant responses.
- **Multi-modal Capabilities**: While earlier versions of GPT were primarily text-based, GPT-3.5 may also have the capability to handle multi-modal inputs, which means it can process both text and other types of data, such as images or structured data.
- **Fine-Tuning**: GPT-3.5 can be fine-tuned on specific tasks, making it adaptable for a wide range of applications. Fine-tuning allows developers to customize the model's behavior for specific use cases, making it more versatile.

4.4 Llama2

Llama2 is different because it has a unique way of finding information and can measure the results. It gives another option instead of using general models. It also uses the Transformer design for understanding and processing language and is meant for specific tasks that require accurate information retrieval. It excels at searching for information databases. It has many parameters usually smaller than GPT-3 and is initially trained to find and organize information. Its downside is that we need a more complicated setup than API-based models.

4.5 GPT2

GPT2, part of the GPT family, is renowned for its versatility and rapid execution, despite its smaller size than GPT-3. It also uses the Transformer architecture for efficient language modeling. It possesses fewer parameters than GPT-3 but is still on the order of several hundred million, remaining relatively lightweight, and is very versatile in its ability to generate text in a variety of contexts. It is initially trained on a large corpus of text and is suitable for text generation, but less specialized in conversational interactions. On the downside, it may lack the contextual specificity needed for extended conversational exchanges.

Key Features:

- **AI-Based Language Model**: Uses machine learning to understand and generate human language.
- **Transformer Architecture**: Employs an advanced method for processing and generating text.
- **Text Generation**: Capable of creating text in various styles and formats.
- **Contextual Understanding**: Effective for basic language comprehension tasks, though limited compared to later versions.

We will observe that applying an exact matching metric to measure the validity of the response compared to our dataset using ChatGPT-2 is not pertinent, as the answers provided are not at all close to the expected ones.

4.6 Pure information-retrieval

We implemented Information Retrieval with DuckDuckGo using DuckDuckGoSearchRun tool from the langchain library to perform a search on DuckDuckGo. This tool sends a query to the search engine and retrieves search results. We provided the search queries to DuckDuckGo, such as "what is a lobster roll." The goal was to obtain relevant information about the topic from the search results. The search results obtained from DuckDuckGo were then processed and the relevant information was extracted. The assumption is that the search results contain information related to the query. Before even using extraction of the answer of DuckDuckGo we took the answer of DuckDuckGoSearchRun, tokenized its answer and compared it to the real answer to see what percentage of exact words were found. The results are seen below. One of its downside is the time taken that was about an hour and a half only to answer 2000 answers which is a long time due to the time needed for each request online.

We then constructed a prompt for an Extractive Question Answering (QA) model using the extracted information. The prompt was designed to present the model with a context formed from the search results and the specific question related to the context. We used DeBERTaV3 from Hugging Face to generate responses based on the constructed prompt. This model is capable of extracting relevant information from the given context to answer questions. The generated answer from the Extractive QA model was compared with the ground truth from the validation set. The performance of the system was evaluated by computing the exact match score. Adjustments to the search query, the extraction process, or the model fine-tuning might be made to improve the overall performance. For example, a good step of pre-processing that we did was to lowercase the sentences and delete punctuation. This approach combines the power of information retrieval from a search engine with the capabilities of a language model for question answering, enabling the system to gather information from the web and generate responses to specific queries. It's an example of leveraging both retrieval-based and generative-based techniques in a coherent workflow. With no fine-tuning, we can see the results below.

4.7 Probabilistic model: BM25 using Pyserini

We developed an intelligent question-answering system that leverages both traditional information retrieval methods (BM25-based document retrieval using Pyserini) and state-of-the-art language models for question answering. We made this model using different components:

The first component was for Document Retrieval using Pyserini to build and index a corpus of documents, here we used Wikipedia. Then we selected DistilBERT a pre-trained question-answering model and did not fine-tune it at first on NQ-OPEN training set. After that, we set up a pipeline using Hugging Face Transformers to perform question-answering on given contexts. Here we combine document retrieval and question answering by using Pyserini to retrieve a set of relevant documents based on user queries. Then we extract relevant information from the retrieved documents and use it as context for the question-answering model. We used the exact match score to measure how well the system's predictions match the ground truth answers. The combination of BM25-based document retrieval and advanced language models enhances the accuracy of the question-answering system. BM25 provides efficient and relevant document retrieval, reducing the context size for question answering. The system can be applied to various domains and datasets, making it versatile for different use cases. One of its downsides is the time taken to answer, indeed for the validation set of 3610 samples, we took 2 hours to answer them all which is about 2 seconds per answer using 5 documents to search for every question. A problem occurs as we observed that varying the number of documents explored also changes the exact match score meaning that it is important to optimize this hyper-parameter and may take some time.

The BM25 score of a document D given a query Q is calculated as follows:

$$\text{Score}(D, Q) = \sum_{t \in Q} \text{IDF}(t) \cdot \frac{f(t, D) \cdot (k_1 + 1)}{f(t, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

where

$$\text{IDF}(t) = \log\left(\frac{N - n(t) + 0.5}{n(t) + 0.5} + 1\right)$$

Definitions:

$\text{Score}(D, Q)$: BM25 score of a document D given a query Q.
t : A term from the query Q.
$\text{IDF}(t)$: Inverse Document Frequency of the term t.
$f(t, D)$: Term frequency, the number of times the term t appears in the document D.
$|D|$: Length of the document D in words.
avgdl : Average document length in the text collection.
N : Total number of documents in the collection.
$n(t)$: Number of documents containing the term t.
k_1 : Tuning parameter controlling the scaling of term frequency.
b : Tuning parameter controlling the degree of document length normalization.

BM25 is a widely used ranking function in the field of information retrieval, particularly in search engines. It is designed to calculate the relevance of documents to a given search query, based on the terms present in both the query and the documents.

The core idea of BM25 is to rank documents based on the frequency and significance of the query terms appearing in them. However, unlike simpler models that might count term frequency linearly, BM25 applies a sophisticated formula that considers several key factors:

1. **Term Frequency (TF)**: BM25 evaluates how many times each query term appears in a document. However, it understands that the relevance does not increase proportionally with term frequency. After a certain point, the addition of the same term contributes less to the document's relevance.

2. **Inverse Document Frequency (IDF)**: This component of BM25 assesses the importance of a term in the entire document collection. Terms that are rare across documents are given higher significance, as their presence in a document could be more indicative of relevance to the query.

3. **Document Length Normalization**: BM25 accounts for the length of documents. It normalizes the term frequency considering the document length, ensuring that longer documents do not inherently have an advantage over shorter ones. This normalization is controlled by parameters in the formula.

The BM25 formula includes two key parameters, k_1 and b, that allow for fine-tuning its behavior:

- k_1 affects how the term frequency is scaled. It controls the extent to which the frequency of a term influences the relevance score.

- b dictates the level of length normalization. A higher value of b gives more weight to document length, while a lower value reduces its impact.

In essence, BM25 provides a balanced and efficient way to rank documents in response to a query, considering both the occurrence and the distribution of query terms in the document collection. Its effectiveness and adaptability to various types of text data make it a cornerstone in modern search technologies.

4.8 Dense indexing with neural models

The integration of dense indexing with neural models in information retrieval and natural language processing (NLP) marks a significant advancement, combining artificial intelligence with sophisticated search techniques. This approach offers a nuanced understanding of text data, transcending traditional keyword-based search methods.

4.8.1 Creation of the Dense Index

1. **Transformation into Dense Vectors**: Documents and queries are transformed into dense vectors using neural models. These vectors encapsulate the semantic features of the texts.
2. **Use of Neural Models**: Advanced neural architectures like BERT, GPT, or their specialized variants encode texts into dense vector representations, capturing linguistic nuances and contextual information more effectively than traditional methods.
3. **Training**: The neural models undergo extensive training on large datasets. This training enables them to generate vector representations that accurately mirror the meaning and contextual nuances of words and phrases.

4.8.2 Functioning of Dense Indexing

1. **Storage of Vectors**: Each document in the collection is represented by a dense vector and stored in a specialized index. This index differs from traditional ones as it maps documents to points in a multidimensional vector space rather than linking specific terms to documents.
2. **Similarity Search**: When a query is made, it's also converted into a dense vector. The search algorithm then identifies documents whose vectors are closest to the query vector in the vector space, often using techniques like Nearest Neighbor Search.

4.8.3 Advantages

1. **Contextual and Semantic Understanding**: Dense indexing facilitates a deep understanding of content, capturing contextual and semantic subtleties.
2. **Handling Polysemy and Synonyms**: It effectively manages polysemy and recognizes synonyms, enhancing the relevance and accuracy of search results.

4.8.4 Challenges and Considerations

1. **Computational Complexity**: Dense indexing demands high computational power and storage capacity.
2. **Updating and Maintenance**: The need for regular updates and retraining of models to adapt to evolving language use adds to its complexity.
3. **Bias and Data Quality**: The performance of neural models can be influenced by the quality and diversity of training data, and biases in data can affect search outcomes.

4.8.5 Conclusion

Dense indexing with neural models offers a sophisticated and effective approach for information retrieval and NLP, adept at handling the complexities of language. However, this technology requires substantial technical resources and ongoing management.

5 Ethical considerations

Using pre-trained models raises some ethical concerns, especially about who is responsible and how the information is shared. We need to think about what is right and fair when developing and using this technology.

- **Bias and fairness:** When we have feelings that make us favor one thing over another and treat everyone in a way that is not based on any advantage or disadvantage. Pre-trained models might have biases from the data they were trained on. This can cause unfair or prejudiced actions. It is very important to find and reduce these biases to make sure the responses are fair.

- **Being accountable and transparent:** Pre-trained models with lots of parameters are hard to understand. It's important to have ways to understand and explain the decisions made by these models, so that we can hold them accountable if they make mistakes or give bad results.
- **Sharing false information:** Pre-trained models can create information that can spread fast. This makes people worry about false information spreading. We need to have ways to check and make sure things are right to reduce these risks.
- **How sensitive the input is:** Some models might be affected by the exact way the questions are asked. Bad people could use this sensitivity to get bad results. We need to make a strong plan and keep checking it regularly.
- **Legal and privacy concerns:** Using pre-made templates could cause legal problems if the answers are wrong. Also, the data used for training needs to be kept secret and private.
- **Making sure everyone can easily use and be a part of something:** Pre-trained models should be available to everyone and not leave out any groups. We need to make sure that models are welcoming to everyone and are adjusted to different cultures and languages.
- **Learning and knowing about something:** It's important to teach people about what these models can and can't do. Understanding how technology works can help us use it in a responsible and ethical way.

Using pre-trained models needs careful ethical consideration to reduce possible dangers. This means making sure that everyone is treated fairly, being honest and open, stopping false information, keeping things private, and making sure everyone can take part. "Developers and users must know they are responsible when using these models in real-life situations."

6 Experimental approaches, Evaluation and Limits

We present here the different results obtained following different methodologies. The first results concern ChatGPT 3.5:

Without fine-tuning

We first try to directly ask the questions without any preparation. Without preparing the prompt, chatgpt's answers are too detailed and in sentence form. So although the answer we are looking for is contained in the ChatGPT output, performance with an exact match metric is very poor. Here is an example of what we get (first line is composed of the question, then the acceptable answers, and the second line is the output from ChatGPT): Figure 2.

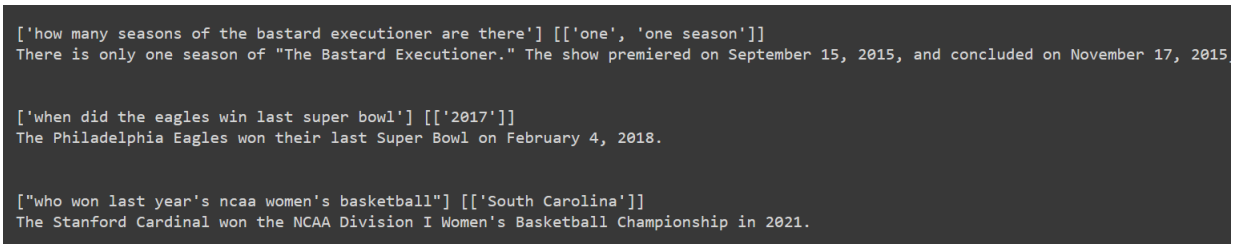

Figure 2: Example of answers from ChatGPT 3.5 without fine-tuning.

Thus, unsurprisingly, we get an overall **score of 0%** using the exact match metric, with a subset of 200 elements from the validation dataset of nq_open

Closed-book

Now before asking the different questions, we perform a work with the prompt to obtain the desired format of answer. We add a system role as an initial dictionary in the list of messages. The

default system role content is “You are a helpful assistant”. But if we edit this, we can actually give our ChatGPT API a different behaviour, depending on the type of answer that we are looking for. We do this via a system role as the first dictionary in the list. Here is our first try to editing the system parameter call (Figure 3):

```
chat_completion = client.chat.completions.create(
    messages=[{"role":'system','content':'''When you reply, go straight to the point.
                                           Use as few words as possible when replying.
                                           Do not even make sentences, just give the relevant words.'''},
```

Figure 3: ChatGPT context modification.

Now the answers that we obtain are much better, in the sense that there are closer to what is expected (Figure 4). With this new context we obtain an **exact match score of 13%** over 200

```
when did the isle of wight become an island ['During the last Ice Age']
After the last ice age.


love yourself by justin bieber is about who ['Rihanna']
self-love


who was the ruler of england in 1616 ['James I']
James I
```

Figure 4: ChatGPT’s answers with our context modification.

elements from the validation dataset of nq_open, which is a good improvement. However, we can still observe a few drawbacks, such as the fact that ChatGPT often put a period at the end of its answer (Figure 5). In such a case the exact match score would be 0 even if the answer is right because the

```
who plays dusty in the movie pure country ['George Strait']
George Strait.
```

Figure 5: ChatGPT’s answer: period problem.

two strings differs by a period. So we add the following sentence to the context : ”Never put a period at the end of your answer”. Unfortunately, ChatGPT still uses periods, even after several attempts to forbid it to use.

Instead of trying to change ChatGPT’s answers by manually changing the context, we suggest now to keep the context we created and to modify some parameters of the exact match score. For instance, it is possible not to take into account some regular expressions, or to remove case sensitivity. We now compute the exact score match ignoring case and punctuation (in the exact_match_metric from hugging face’s library ”evaluate”, we set ”ignore_case” and ”ignore_punctuation” parameters to True). With these specifications, we now obtain an exact match score of: **27%**.

In ChatGPT, instead of trying to influence its answers by detailing in the context what type of answers we want, we are now using the training part of the nq_open dataset as a context (question/answer) in order to fine-tune it to get the desired format of answers.

We can finally compare in this case the three metrics used in the previous part : exact match score, exact match score ignoring case and punctuation, and similarity percentage score.
As a comparison, with ChatGPT2, we obtained an exact match score of **: 7%**

count	2030.000000
mean	0.440977
std	0.428545
min	0.000000
25%	0.000000
50%	0.400000
75%	1.000000
max	1.000000

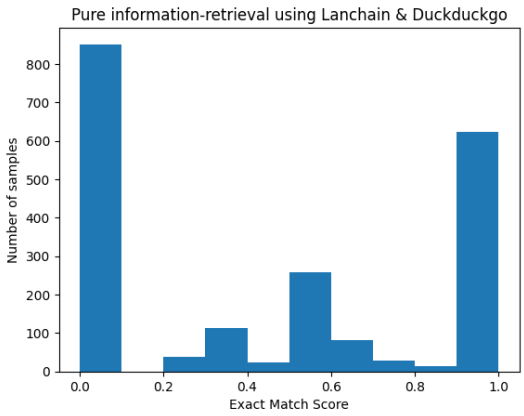

Figure 6: Histogram and description of the basic results using only the percentage of words from the ground truth answer found in the DuckDuckGoSearchRun

count	3600.000000
mean	0.223120
std	0.380130
min	0.000000
25%	0.000000
50%	0.000000
75%	0.333333
max	1.000000

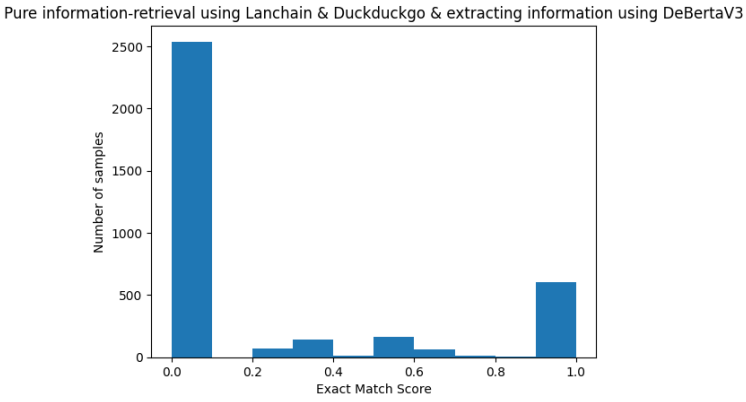

Figure 7: Histogram and description of Pure Information Retrieval Exact-Match using DuckDuckGoSearchRun and DeBERTaV3 for answer extraction

count	3608.000000
mean	0.151668
std	0.326608
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	1.000000

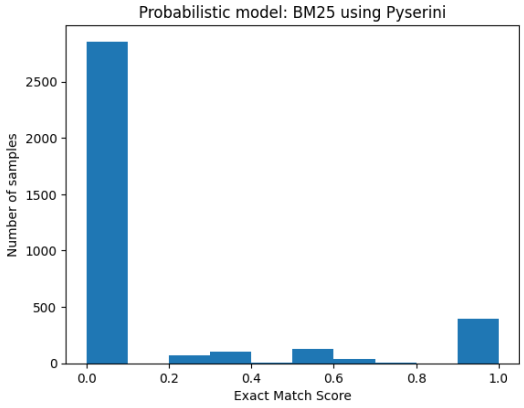

Figure 8: Probabilistic model-BM25 using Pyserini and DistilBERT base cased distilled SQuAD for answer extraction

7 Conclusion

7.1 Experimental Approaches and Results

We employed various experimental methodologies, including closed-book question answering with models like ChatGPT and GPT2, information retrieval using external search engines, and probabilistic modeling with BM25. Our results revealed significant variations in performance across different approaches.

Closed-book QA: While closed-book QA using language models showed promise, achieving an exact match score of 7% and 13% for GPT2 and ChtGPT 3.5 respectively, there were challenges with model output consistency and fine-tuning requirements.

Information Retrieval: Utilizing external search engines for information retrieval yielded mixed results, with mean scores ranging of 44% without information retrieval and 22% with showing a loss of information.

Probabilistic Modeling with BM25: Implementing the BM25 model using Pyserini demonstrated moderate performance, with a mean score of 15% for information retrieval tasks, indicating room for improvement but also the need to choose the number of documents which will expend the time taken for each answer and the introduction of noise in the added information.

7.2 Ethical Considerations

Throughout our exploration, we remained cognizant of the ethical implications inherent in using pre-trained models and conducting large-scale experiments. We addressed concerns such as bias mitigation, accountability, privacy, accessibility, and the dissemination of false information, emphasizing the importance of responsible AI development and deployment.

7.3 Conclusion and Future Directions

In conclusion, our study sheds light on the multifaceted nature of question answering research, highlighting both the advancements made and the challenges that lie ahead. Moving forward, several avenues for improvement and future research emerge:

Temporal Aspects: Explore methods for incorporating temporal dynamics into question answering systems to account for evolving information over time. Indeed ChatGPT and GPT2 were trained at a fix point in time using the information available at the time. So they are not aware of the latest news and cannot answer any questions about recent events.

Document Management: Investigate strategies for efficiently managing the number and size of documents in question answering contexts, optimizing performance without sacrificing relevance.

Balancing Knowledge Sources: Strike a balance between leveraging the rich knowledge encoded in language models and integrating insights from external indices or databases to enhance the robustness and accuracy of question answering systems.

Fine-tuning and Adaptability: Experiment with fine-tuning techniques to adapt pre-trained models to specific domains or use cases, maximizing performance and relevance in diverse contexts.

In summary, while our study provides valuable insights into the current state of question answering research, there is still much work to be done to advance the field and address emerging challenges. By embracing interdisciplinary approaches, fostering collaboration, and upholding ethical principles, we can pave the way for more sophisticated and inclusive question answering systems in the future.

8 Bibliography

References

[1] Alec et al. Improving language understanding by generative pretraining. pages 1–16, June 2018.

[2] Alec Radford et al. Language models are few-shot learners. pages 1–15, June 2019.

[3] Ashish Vaswani et al. Attention is all you need. pages 1–15, 2017.

[4] Brown et al. Language models are few-shot learners. 2020.

[5] Devlin et al. Bert: Pre-training of deep bidirectional transformers for language understanding. pages 5992–6002, 2018.

[6] Hugo Touvron et al. Llama 2: Open foundation and fine-tuned chat models. 2023.