Developing and Assessing a Question Answering System in Natural Language Processing

Amalia-Maria Postolache, Irina Rebegea

January 2024

1 Abstract

This paper presents the design, development, and evaluation of a Question Answering system engineered to excel in two critical tasks: identifying correct long answers and extracting concise short answers from the identified long responses. The study also explores the impact of pretraining on the system's performance, comparing two deep learning models with pretraining, one leveraging GloVe embeddings and the other combining both GloVe and BERT, against one without. The performance analysis involved metrics such as accuracy, precision, recall, and F1 score. The pretraining-based solutions demonstrated a substantial boost in performance compared to the baseline solution, showcasing the effectiveness of incorporating external knowledge in the model architecture.

2 Introduction

2.1 Kaggle Competition Overview

The research presented in this paper was conducted as part of the TensorFlow 2.0 Question Answering Kaggle competition. The primary objective of the competition was to tackle the complexities associated with understanding and extracting information from large text composition. Participants were provided with a dataset comprising a collection of documents, each associated with a set of user queries. The challenge lay in developing a QA system that could accurately identify the correct long answers within the documents and subsequently extract concise short answers relevant to the user queries.

2.2 Approach Overview

Our approach to addressing this problem involved a combination of natural language processing techniques, and strategic preprocessing steps. To ease the methodology of the system's developing process, we focused on the two key components of the problem:

- Long answer identification task - building a model capable of comprehending the context of a given query within a document and pinpointing the most relevant long answer.
- Short answer identification task - extracting short, concise responses from the long answers that have been previously identified.

In terms of the competition's metric, short and long answer formats do not receive separate scores, but are instead combined into a micro F1 score across both formats. Also, the metric does not use confidence scores to find an optimal threshold for predictions. As mentioned earlier, the initial model has also been pretrained in order to observe the difference in the system's performance on the same competition dataset.

3 Review of related works

3.1 QA Systems Introduction

Question Answering (QA) stands as a specialized domain within the realm of Information Retrieval (IR). Functioning as a man-machine communication device, the fundamental concept of QA systems in Natural Language Processing (NLP) is to provide relevant answers in response to questions proposed in natural language. A typical QA system comprises three primary modules, each featuring a core component alongside additional supplementary elements. These modules are the "Query Processing Module," centered around question classification; the "Document Processing Module," focused on information retrieval; and the "Answer Processing Module," centered around answer extraction. In the "Question Processing Module," the system identifies the question's focus, classifies its type, determines the expected answer type, and reformulates the question into semantically equivalent variations. Information retrieval (IR) system recall is very important for question answering, because if no correct answers are present in a document, no further processing could be carried out to find an answer. The conclusive element in a question-answering system is answer extraction, setting it apart from conventional text retrieval systems.
The technology employed for answer extraction plays a pivotal and decisive role in shaping the final outcomes of a question-answering system. As such, answer extraction technology is recognized as a module within the broader framework of the question-answering system.

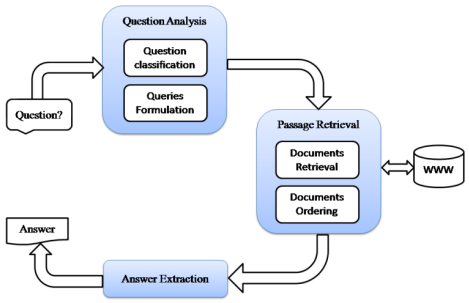

Figure 1: Architecture of a QA system

3.2 Literature review

In the early stages of question-answering systems, one notable example was BASEBALL, a program developed by Green *et al.* in 1961 to respond to inquiries about baseball games played in the American league for a single season. Another significant early endeavor was the LUNAR system (1971), created following the Apollo moon mission to facilitate lunar geologists in accessing, comparing, and evaluating chemical analysis data on lunar rock and soil composition. Additional systems such as SYNTHEX, LIFER, and PLANES pursued the common objective of providing answers to questions posed in natural language.

QAS features		appear in	Used techniques
NLIDB:(Natural Language Interfaces to Databases)		PRECISE [9]	Identifying classes of questions
		The formal semantic approach [10]	Intermediate representation language
		MASQUE/SQL [11]	Portable NL front end to SQL databases
		BASEBALL [8] [12]	Specific domain Systems
Open Domain Question Answering	Document-based Question	[13], [14], [16], LASSO [15]	Deep linguistic analysis and iterative strategy
		FALCON [17]	Hierarchies of question types based on the types of answers sought

Figure 2: Popular QA features and techniques

Despite these early developments, question-answering systems have evolved significantly over the ensuing decades, culminating in the contemporary structure we observe today. As mentioned previously, modern QA systems are characterized by a three-part foundation: question classification, information retrieval, and answer extraction. Consequently, each of these components has become a focal point for researchers in the field of question answering.

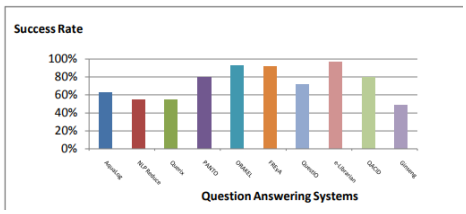

Figure 3: Performance results of popular QA systems

3.3 TensorFlow 2.0 Question Answering - results

Most of the models that participated in the competition were pretrained, due to their better performance. Several submissions leveraged the BERT baseline model, with some of them achieving silver or bronze rankings in the competition. In the following figure we can observe the top ten rankings in the competition for the private leaderboard.

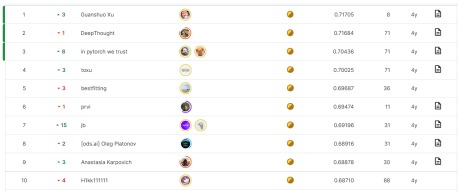

Figure 4: TensorFlow 2.0 Question Answering - private leaderboard

The top performer's solution of the competition on the private leaderboard diverged from the baseline paper by training on provided candidates instead of sampling from original documents, utilizing a total of 40 million candidates in the training data. Model architecture mirrored the baseline paper, featuring a 5-class classification branch and a 2-span classification branch, and the final submission was an ensemble of one Bert-base, two Bert-large (WWM), and two Albert-xxl (v2) models, all uncased. The solution that secured the second place on the leaderboard uses custom heads and a BERT transformers backbone for a single TensorFlow 2.0 model. For modeling and training, the system leverages the transformers library.

4 QA System Development

The QA system is designed to answer questions based on a given dataset containing questions, document texts, and annotations. It aims to provide accurate answers to questions by first classifying long answer candidates and then extracting relevant short answers. The use of machine learning techniques, including tokenization and embedding (topics which will be discussed in the next section), enhances the model's understanding of the input data.

4.1 Benchmark Performance

Our solution focuses on the two primary tasks mentioned earlier, long answers identification and short answers extraction, so we build two models, one for each task. After loading 10000 training instances and 500 validation questions from the competition input files, the testing dataset is preprocessed in order to extract information from the long answer candidates, such as the content of the response and the indices where the response is found in the document. We select these candidates with a sample rate of 15. Preprocessing functions, such as *remove_stopwords*, and *remove_html* are used for text cleaning. In order to create a numeric array representing indices for the unique words from the dataframe, we use the TextVectorization layer from TensorFlow to tokenize the input data, following up with the embedding process. At this step, we create an embedding layer that learns word embeddings during training.

The classification model for the long answers identification task is designed to predict whether a given long answer candidate is correct or not. The model is trained using the provided training data with class weights to handle class imbalance, its architecture including embedding layers, convolutional layers, and dense layers. We also dynamically adjust the learning rate during training based on the validation loss.

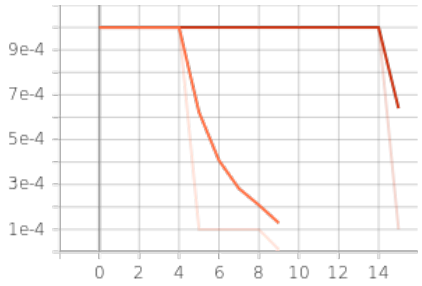

Figure 5: Learning rate adjustment during training

After the model predictions are generated, functions to filter and process the test and validation data based on model prediction probabilities are used.

Moving our focus on the short answers extraction task, we are using the annotations to extract short answer information. Similarly, tokenization and

encoding are performed for short answers. A separate model is defined for predicting the start and end indices of short answers. After training and evaluating the model, the test data is updated with the predicted short answer indices, after they were decoded from model predictions back to token indices.

The .csv file containing the predicted long and short answers tokens is created for submission.

The models are evaluated on the validation dataset using metrics like accuracy, recall, precision, and F1 score. Subsequently, we will analyze the performance of the two models, both individually and together. Hyperparameter tuning is conducted for each model to identify the most suitable configuration, involving variations of the batch size and optimizer. The obtained values for each combination of the first model, with a fixed number of epochs set to 5, are detailed in table 1.

Optimizer	**Batch size**	**Accuracy**	**Precision**	**Recall**	**F1**
Adam	64	0.9207	0.3444	0.4730	0.3986
Adam	128	0.9129	0.3156	0.5060	0.3888
Adam	256	0.9264	0.3425	0.4302	0.3814
SGD	64	0.7302	0.1511	0.8945	0.2586
SGD	128	0.6838	0.1153	0.9078	0.2046
SGD	256	0.7011	0.1348	0.8849	0.2339

Table 1: Hyperparameter tuning on the first model

Due to a high number of false positives in answering questions, we obtain inflated values for accuracy, which do not entirely reflect the model's performance, as it might classify all responses as false. For this reason, we use precision and F1 score to choose one model for each optimizer.

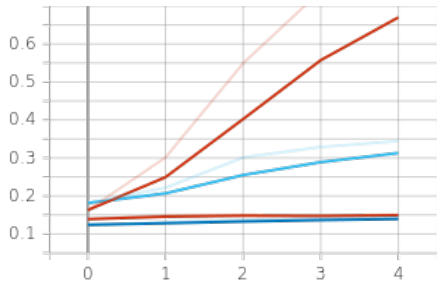

Figure 6: Train and validation precision for models Adam-64 and SGD-64 (upper red line - Adam train, upper blue line - Adam validation, lower red line - SGD train, lower blue line - SGD validation)

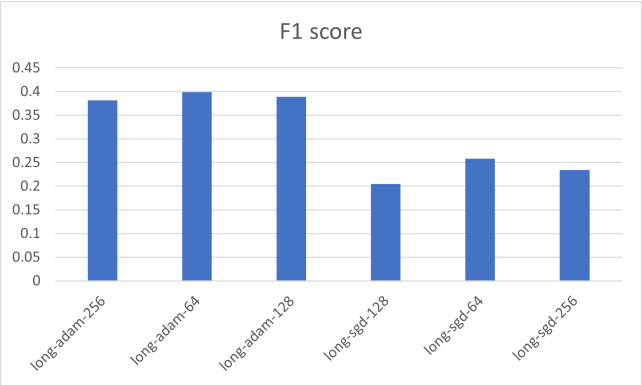

Figure 7: F1 score for the configurations of the first model

Analyzing figures 6 and 7, we conclude that for the first model the combination of Adam optimizer and batch size 64 yields the best results. We use the same procedure for the second model. Thus, we can view its results in table 2, with a fixed number of epochs set to 10.

Optimizer	**Batch size**	**Precision**	**Recall**	**F1**
Adam	32	0.2521	0.3333	0.2620
Adam	64	0.2342	0.3235	0.2428
Adam	128	0.1925	0.3843	0.2326
SGD	32	0.1446	0.3803	0.2096

Table 2: Hyperparameter tuning on the second model

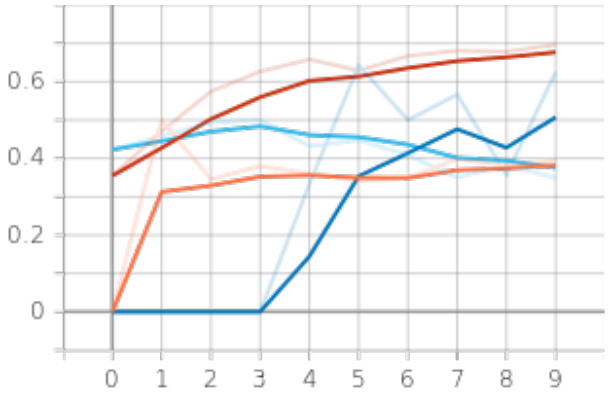

Figure 8: Train and validation start token precision for models Adam-64 and SGD-32 (red line - Adam train, light blue line - Adam validation, orange line - SGD train, dark blue line - SGD validation)

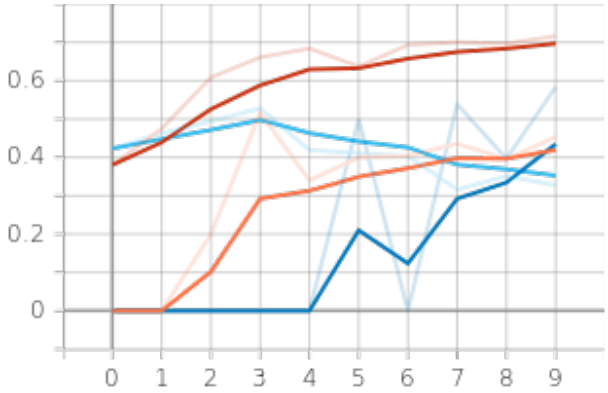

Figure 9: Train and validation end token precision for models Adam-64 and SGD-32 (red line - Adam train, light blue line - Adam validation, orange line - SGD train, dark blue line - SGD validation)

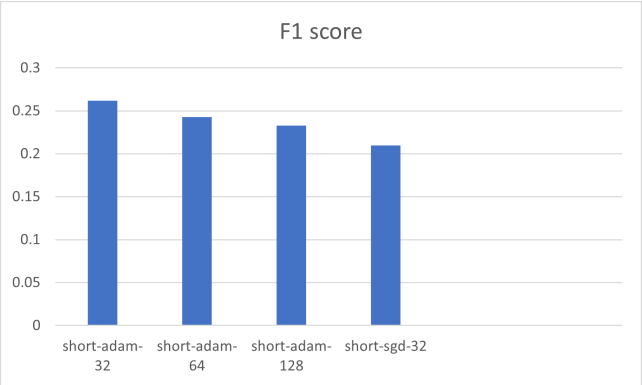

Figure 10: F1 score for the configurations of the second model

Based on figures 8, 9 and 10, we infer that for the second model the combination of Adam optimizer with either batch size 32 or batch size 128 is the best, depending whether we want to focus on precision or F1 score. Combining the best choices from both models, we obtain the following results:

Precision	**Recall**	**F1**
0.3134	0.3676	0.3349

Table 3: Results of the baseline model

4.2 Ablation Study

In order to improve the performance of the prediction task, we have also created two different QA models that use pretraining. The overall workflow involves the steps mentioned for the basic model, but the numerical representation of the words is no longer learned by the models; instead, they retrieve embeddings from pretrained models, specifically GloVe for the first model and a combination of BERT and GloVe for the second model. Both these models achieve similar results, which outperform the base solution in terms of precision, recall and F1 score.

Model	**Precision**	**Recall**	**F1**
GloVe	0.41107	0.629	0.4968
BERT	0.4219	0.608	0.4977

Table 4: Results of the pretrained models

At this stage, we conducted experiments by varying the sizes of the training and validation datasets, increasing the dimension of the vocabulary, extending the number of training epochs, and incorporating regularization techniques into the model to mitigate overfitting. Despite these efforts, the results did not demonstrate improvement. For the model which uses GloVe we also performed hyperparameter tuning, and the results can be seen in tables 5 and 6.

Optimizer	**Batch size**	**Accuracy**	**Precision**	**Recall**	**F1**
Adam	64	0.8725	0.2402	0.7092	0.3589
Adam	128	0.8649	0.2216	0.7468	0.3418
Adam	256	0.8728	0.2258	0.7085	0.3796
SGD	64	0.7932	0.1847	0.8056	0.3006
SGD	128	0.8195	0.1732	0.8149	0.2733
SGD	256	0.7732	0.1656	0.8739	0.2784

Table 5: Hyperparameter tuning on the first pretrained model

Optimizer	**Batch size**	**Precision**	**Recall**	**F1**
Adam	32	0.1462	0.3280	0.1874
Adam	64	0.1548	0.3482	0.2185
Adam	128	0.1319	0.3120	0.1957
SGD	32	0.0926	0.3043	0.1420
SGD	64	0.0878	0.2964	0.1355

Table 6: Hyperparameter tuning on the second pretrained model

As before, we focus on precision and F1 scores to determine the best performing model. Based on figures 11 and 13, we conclude that for the first model, the combination of Adam optimizer with batch size 256 performs best. Similarly, in figures 12, 14 and 15, we see that Adam with batch size 64 is better than SGD with batch size 32.

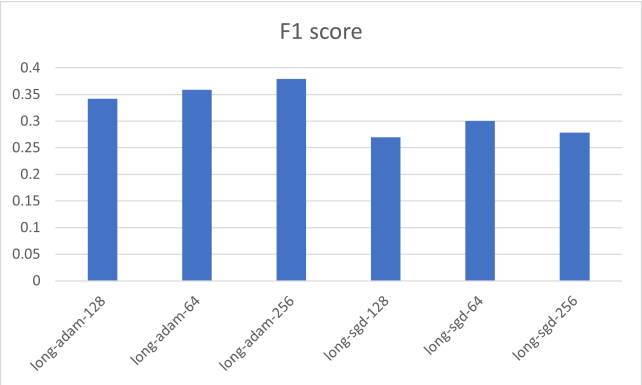

Figure 11: F1 score for the configurations of the first pretrained model

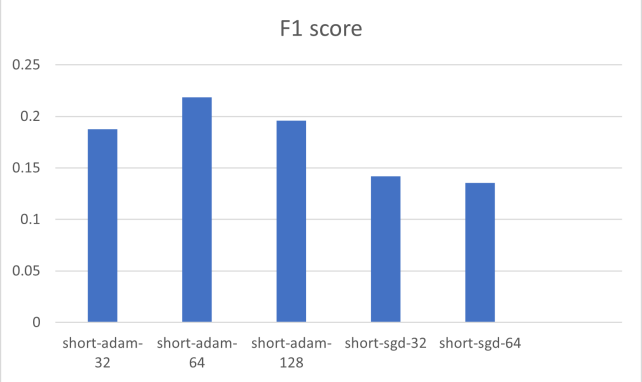

Figure 12: F1 score for the configurations of the second pretrained model

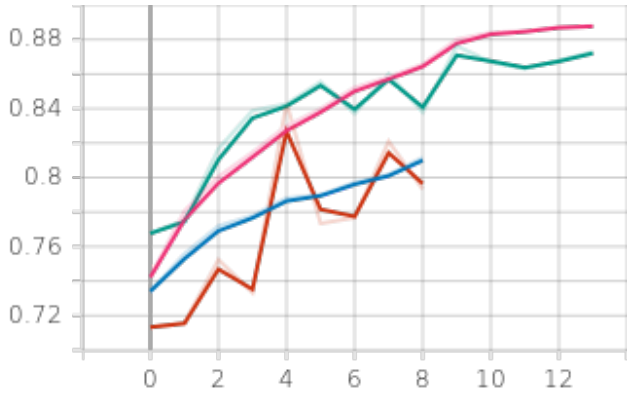

Figure 13: Train and validation precision for models Adam-256 and SGD-64 (pink line - Adam train, green line - Adam validation, blue line - SGD train, red line - SGD validation)

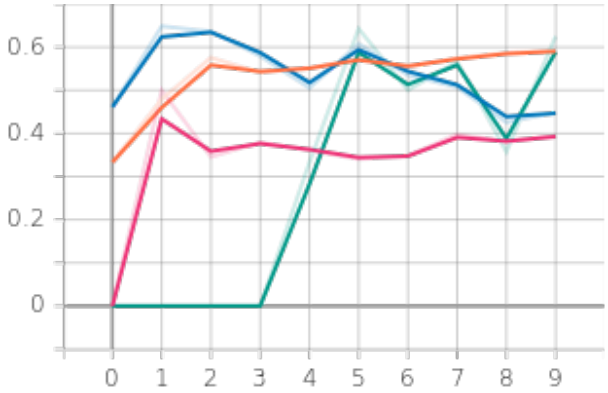

Figure 14: Train and validation start token precision precision for models Adam-64 and SGD-32 (orange line - Adam train, green line - Adam validation, blue line - SGD train, pink line - SGD validation)

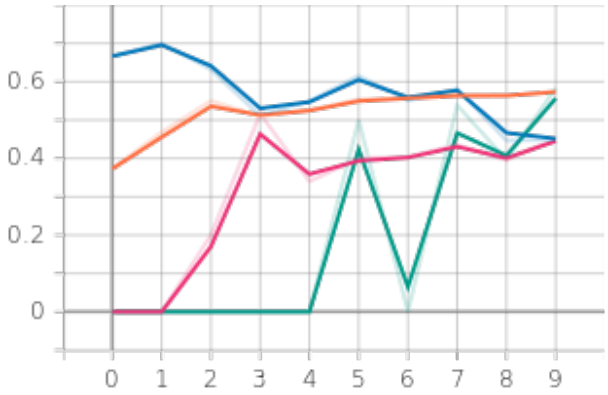

Figure 15: Train and validation end token precision precision for models Adam-64 and SGD-32 (orange line - Adam train, green line - Adam validation, blue line - SGD train, pink line - SGD validation)

Next, we will analyze the performance of the methods on metrics such as speed, compute and memory usage.

Solution	**CPU (%)**	**RAM (GB)**	**Disk (GB)**	**Time (s)**
Baseline	337%	5.4	4.3	7800
GloVe	350%	3	4.3	5400
BERT	-	-	-	8100

Table 7: Performance Metrics for different solutions ran on CPU

The baseline solution exhibits high CPU usage, indicating potential multi-threading or parallel processing. The RAM usage is relatively high at 5.4 GB, and the execution time is the longest at 7800 seconds. Considerable computational resources are utilized, and the overall performance may benefit from optimization.

The GloVe-based solution shows a similar trend in high CPU usage but with a slightly increased RAM efficiency at 3 GB. Despite the lower CPU usage compared to the baseline, the execution time is significantly reduced to 5400 seconds. This suggests that leveraging GloVe embeddings contributes to improved efficiency, resulting in a faster execution time.

The BERT solution, executed on GPU, exhibits a completion time of 8100 seconds. Comparatively, both the Baseline and GloVe solutions, executed on CPU, achieve faster completion times of 7800 and 5400 seconds, respectively. Given that the BERT solution on GPU does not outperform the CPU-based solutions in terms of execution time, and its results are similar to the GloVe solution, it raises the consideration that running BERT on GPU might not provide a significant advantage in this particular context.

The predictions made by the GloVe-based solution were submitted to the competition, obtaining a score of 0.12.

5 Discussion

We consider the proper processing of data as the most crucial factor influencing the model's performance. The dataset comprises data in HTML format, and while we have removed HTML tags, there is an option to train the model in order to learn them. Another challenge lies in instances with excessively long answers that exceed the segment length. We have yet to address the scenario where short answers may be exclusively of the "yes/no" type. Additionally, we may have not tackled properly the imbalance introduced by the substantial number of negative responses to questions. To address data augmentation, we could have considered shuffling sentences within paragraphs. Despite these challenges, we acknowledge the importance of refining data processing techniques to further enhance model performance.

6 Conclusion

In this report, we have analyzed the performance of deep learning methods to address the question answering problem, where the task involves identifying long answers to questions from a set of possible responses and extracting short answers from the correct long answers. We implemented a baseline solution that does not employ pretraining or additional data, resulting in a competition score of 0.06. Additionally, we developed two pretraining-based solutions, one utilizing GloVe embeddings and the other incorporating both GloVe and BERT. The GloVe-based solution achieved a score of 0.12. Performance analysis of the methods was conducted using metrics within the notebook, including accuracy, precision, recall, and F1 score. The baseline solution achieved an F1 score of 0.3, while the two pretraining-based solutions obtained approximately 0.5 F1 score, demonstrating notable improvement in performance.

References

[1] Ali Mohamed Nabil Allam and Mohamed Hassan Haggag, *The Question Answering Systems: A Survey*, International Journal of Research and Reviews in Information Sciences (IJRRIS), Vol. 2, No. 3, September 2012 https://www.researchgate.net/profile/Ali-Allam-4/publication/311425566_The_Question_Answering_Systems_A_Survey/links/5845873808ae8e63e62862b1/The-Question-Answering-Systems-A-Survey.pdf

[2] R.Mervin, *An Overview of Question Answering System* , International Journal Of Research In Advance Technology In Engineering (IJRATE), Vol 1, Special Issue, October 2013 https://www.researchgate.net/profile/R-Mervin/publication/320978810_An_Overview_of_Question_Answering_System/links/5a055663458515eddb84f105/An-Overview-of-Question-Answering-System.pdf

[3] Amit Mishra, Sanjay Kumar Jain, *A survey on question answering systems with classification*, Journal of King Saud University - Computer and Information Sciences, Vol 28, Issue 3, 2016, https://www.sciencedirect.com/science/article/pii/S1319157815000890#b0095

[4] Abdelghani Bouziane, Djelloul Bouchiha, Noureddine Doumi, Mimoun Malki, *Question Answering Systems: Survey and Trends*, Procedia Computer Science, Vol 73, 2015, https://www.sciencedirect.com/science/article/pii/S1877050915034663

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, https://arxiv.org/abs/1810.04805

[6] Jeffrey Pennington, Richard Socher, Christopher D. Manning, *GloVe: Global Vectors for Word Representation*, https://nlp.stanford.edu/pubs/glove.pdf

[7] Tom Kwiatkowski et al., *Natural Questions: a Benchmark for Question Answering Research*, https://research.google/pubs/natural-questions-a-benchmark-for-question-answering-research/