# Conversational Agent Embodying a Historical Figure using Transformers

**Alexandru Dima**
University Politehnica of Bucharest
313 Splaiul Independetei, Bucharest, Romania
alexdima98@gmail.com

**Traian Rebedea**
University Politehnica of Bucharest
313 Splaiul Independetei, Bucharest, Romania
traian.rebedea@cs.pub.ro

**ABSTRACT**
In this paper, we discuss how transformers, the current state of the art solution for Question Answering (QA) can be applied for implementing a conversational agent that can offer accurate and informative answers to questions about any user-chosen historical figure. We use Wikipedia to obtain data for every personality chosen and an information retrieval engine to store the data and perform full-text queries for obtaining the context dynamically. We tested the performance and accuracy for the different transformer models, data store, and retrieval methods. The best result was obtained using an ALBERT model with a heuristic retrieval method. This solution obtained an F1 score of 66.06 and an Exact Match (EM) score of 42.22.

**Author Keywords**
Conversational agent; natural language processing; question answering; data retrieval; transformers.

**ACM Classification Keywords**
I.2.7 Natural Language Processing

DOI: 10.37789/rochi.2021.1.1.3

## INTRODUCTION

We live in an age where information is easily accessible for anyone that has access to the internet. All one needs is a smart device (smartphone, tablet or pc) with access to an internet connection and they can search for almost any desired topic. As of 2021, 5.16 billion people (over 65% of the global population) have access to the internet. As a comparison, in 2011 there were 2.26 billion internet users (about 32% of the world population at that time) and in 2001 there were 0.51 billion users (8.6% of the world population) [7]. It should also be noted that the number of websites on the internet has also seen steady growth over the years, reaching 1.72 billion websites in 2019 (as shown in figure 1).

As mentioned, the ease of accessing information has grown significantly over the years, but because of this growth, the difficulty of finding relevant and veridical information has also increased. One of the reasons misinformation can spread so easily is the growth of the online environment and the nature of human beings. Fake news (misinformation) is created by manipulative people who use the digital environment to take advantage of humans' inborn preference for comfort, convenience and their craving for answers that suit their beliefs [9].

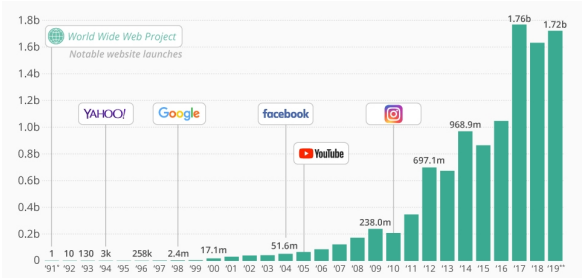



**Figure 1: Evolution of number of websites [3]**

We also need to mention that, in their search towards finding the answer to the questions they pose, users tend to struggle to navigate the wealth of online information that is now available [4].

To be able to mitigate misinformation, but still accommodate humans' desire for knowledge, the need for an automated question answering system becomes ever more urgent. Unlike search engines which can only return a ranked list of documents that may or may not contain the answers users are pursuing, we need a system that can quickly respond to the questions users pose with sufficient context to validate the answer [4]. By using such a system, and carefully selecting the sources that the system (the question answering agent) uses, we can ensure that users can quickly identify the genuine answers to their questions. Therefore, the creation of such systems, not only will provide an easier alternative than search engines for finding information online but will also be

[3] Number of Online Websites from 1991 to 2019. Accessed June 02, 2021. URL: https://www.statista.com/chart/19058/how-many-websites-are-there/

able to attenuate the negative impact that fake news has created in the online environment.

One beneficial domain where such a question-answering system would be useful is history. History is essential for people to understand themselves and the world around them. There is important history for every field and topic from medicine and information technology to music and art.

The study of history allows an individual to comprehend more about the various aspects of our world such as technology, governmental systems and the link between cause and effect. One of the best ways to understand present-day issues is by analyzing the cause-and-effect chain that led to the way things stand in the present [10]. To also make the experience more enjoyable to the user, the best way for him to experience history is by interacting with someone who lived in that context. For that purpose, a question-answering agent that embodies a historical figure is a good approach for resolving this problem.

This paper is organized as follows: firstly, we discuss some existing approaches on building conversational agents for historical figures, after we describe our solution and its results and finally, we will present our conclusions.

## RELATED WORK

There were a few prior approaches to building a conversational agent that embodies a certain personality. In this section, we will present these approaches and highlight the differences between them and our approach. The first approach was published by Haller and Rebedea [3], where they built a rule-based question answering conversational agent. They gathered information about the historical figure and saved that information as a fact (a triplet comprised of a subject, an action or a verb, and the object). We can see an example of how a fact looked in figure 2.

"People described me as a neurotic psychopath"

(fact: (me, health-opinion, neurotic_psychopath))

**Figure 2: Fact example**

In this example, "me" is the subject, "health-opinion" is the action and "neurotic-psychopath is the object". They saved all the facts in the knowledge base of a chat-bot engine called ChatScript. Using this engine, they constructed patterns for the engine to use when generating the answer for each question.

The second approach was made by Bogatu et al. [1] who built a rule-based QA agent, but unlike the previous approach, this used two methods to find the necessary information to answer a question. The first method used an ontology (an explicit specification of a simplified view someone wants to represent) to build their knowledge base. Unlike the previous attempt where the pattern matching was made on the facts, here they also considered the meanings the word has. If this method fails in providing an answer, they attempt to find a sentence that best answers the given question from a set of candidate questions. The sentence selection method has three filters to eliminate bad sentences. The first one eliminates the sentences that do not match the type of answer the question needs (if the question is about when someone was born, the answer needs to be a date). The second one eliminates sentences that do not preserve the semantic relationships between constituents that appear in the question. The final filter generated more questions with different formulations but the same meaning to increase the chance of finding a good answer. To generate the answers, they also used ChatScript and pattern matching. This approach obtained an approximate accuracy of 40% for answering the question.

A more recent approach was proposed by Ilie and Rebedea [5] using a sequence-to-sequence model with attention. The encoder and decoder were composed of three layers of 256 GRUs. Is important to mention that their decoder also used beam search, a technique that at each step in the decoding process, keeps track of the best k candidates for the decoding. Unlike the previous attempts which focused more on QA, this attempt focused on the dialogue part, creating an agent that is good at impersonating a given figure. To assess the quality of the response they used metrics for MT tasks like BLUE score which measures the similarity between the target utterance and the candidate utterance, but also tested the agent with human subjects.

Similar two the first two models, the agent developed for this project will focus on the QA task, rather than the impersonation part. The important differences between this project and the previous approaches are that the agent in this project uses the transformer model to generate the answers to the question posed by the user and that, we will use metrics for assessing the accuracy and performance of the agent.

## PROPOSED SOLUTION

Our proposed solution has two main components: a backend that handles all the logic for data retrieval and interaction with the transformer models and a frontend for the user to be able to easily access the solution. In this paper we will focus only on the backend.

### General Architecture

The backend is comprised of three main modules (figure 3). Each module was constructed to serve a single purpose, for creating a well-defined separation between all the elements that form the backend.

The backend controller (which is implemented in app.py), the module for interaction with Elasticsearch (which is implemented in the class *ElasticsearchPersonalities* from Elasticsearch_controller.py) and the data retrieval module for personalities (implemented in the *PersonalityData* class from download_personality_data.py).

For interacting with the data stored in Elasticsearch the backend controller uses the ElasticsearchPersonalities. Sequentially, this class uses in one of its methods the PersonalityData class for extracting all necessary

information about a given personality from the web, more exactly from Wikipedia.

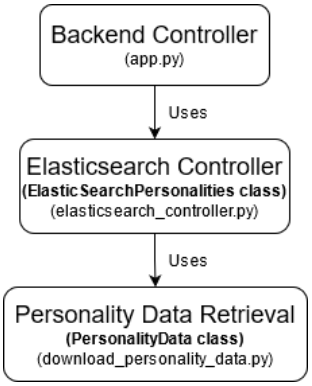



**Figure 3: General architecture**

### Personality Data Retrieval

Because a transformer model needs a context from which to predict the answer to the question posed by the user, the PersonalityData class (figure 4) is an important part of the backend.

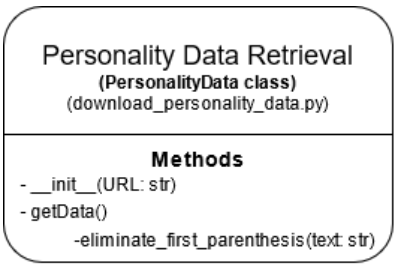
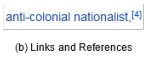



**Figure 4: Personalitydata class**

In this version of the software, we will use Wikipedia as the source of data for every historical figure that will be added to the application. This class downloads all important data that can be found on the Wikipedia article of a selected historical figure. Since we are only interested in the text data that can be found on a page, this class needs to filter all the available data to obtain useful information. As seen in figure 5, the other types of data stored on a Wikipedia page (except text data) are tables, links and references, pronunciation and spellings, pictures and their labels, external references and others.


(a) Table




(b) Links and References




(c) Pronunciations

(d) Pictures and their Labels

**Figure 5: Useless types of data on Wikipedia for a text retrieval QA system**

We used BeautifulSoup not only to obtain all useful paragraphs and their headings from Wikipedia but also to filter out useless information from the paragraphs based on the HTML tags and their ids and classes. Since most personality pages on Wikipedia have in the first paragraph a parenthesis that contains information already detailed in the rest of the article, the method eliminate_first_parenthesis was used to remove it.

This class returns the processed information in three formats: topics, paragraphs, and phrases. The topics are subchapters on the Wikipedia page, they usually contain multiple paragraphs that were grouped on the heading that appeared before them. Paragraphs are all the processed paragraphs obtained and phrases are all the phrases that have been extracted from the paragraphs using Python's natural language toolkit (NLTK) package.

### Elasticsearch Controller

This controller is used for storing the data returned from the personality data retrieval module in an opened Elasticsearch instance and to retrieve it in different ways for the prediction of the answer.

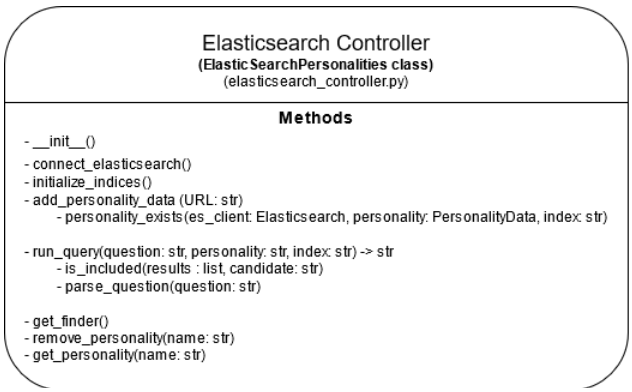



**Figure 6: Elasticsearch controller**

When instanced the ElasticsearchPersonalities class will first try to connect to an Elasticsearch instance and then initialize the indices (an index can be viewed as a database). Needless to say, if there is not an opened Elasticsearch instance then the application will not function. The indices available in this application can be seen in figure 7.

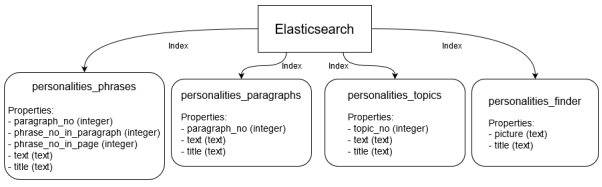


**Figure 7: Elasticsearch Indices**

The most important methods in this class are add_personality_data and run_query. The first uses the PersonalityData class to obtain information about the personality from the given URL parameter (this should be the URL to the Wikipedia page of the desired historical figure). After the data has been received, it tries to add it in each index from figure 7 (this process is called indexing). The personality_exists function will check if the data for the associated historical figure already exists in that index so that it will not be added it again.

The run_query method is used for dynamically extracting the context that will be used for the prediction of the transformer. The reason why we can not use the whole Wikipedia page in the prediction is that transformers need significant dedicated RAM to function. The dedicated memory required and the time needed to predict the answer to the question increase proportionally to the size of the context. In addition, even though the attention mechanism of the transformer allows it to be able to use a large quantity of text, if that quantity becomes too massive it can negatively impact the performance of the transformer (more about the concrete impact and different results can be found when we will discuss the results). Because of these reasons, we need a way to dynamically find the context that should contain the answer to the question posed by the user.

The run_query method first uses the parse_question function to modify the question to a format that yields better results when used as an Elasticsearch query. It has three steps, firstly it eliminates leading and trailing whitespace in the question, then it removes all the punctuation in the question and lastly, it eliminates all wh-pronouns (these are the pronouns that are always present in a question, like “why”, “who”, etc.). We can see an example of how this works in table 1.

| | |
|---|---|
| Original Question | At what subjects was Einstein good in school? |
| Processed Question | At subjects was Einstein good in school |

**Table 1: Example for processed question**

The run_query method has four ways of returning the probable context that the transformer will use. If the retrieval method is one of *phrases, paragraphs* or *topics*, it will run a query using the personality name and parsed question and return several results (this number depends on the retrieval method and it was selected following the results obtained, more on this when we discuss the results). Then it will concatenate all the results in a string and return the string to be used as context. The last retrieval method is heuristic. It uses a heuristic created using the results obtained to return using multiple queries, a different number of results from each of the three previous indices. After each of the three queries, the results are concatenated in a list. When concatenating the results, we used the is_included function to remove the duplicates.

### Backend Controller

Most of the functions that are defined in this component are functions that interact with the ElasticsearchPersonalities object and return the result as JSON. This component uses Flask to create a web API that the frontend uses to interact with the transformer model and Elasticsearch.

For interacting with the available transformer models, we will use Hugging Face’s transformers package. With this package we can easily import any transformer model available in their database and use it for prediction using their pipeline package. We only need to create a QA pipeline using the desired transformer model and then we can use the pipeline for predicting the answer to a question.

The only route that needs to be discussed is */api/question*. When receiving a valid request on this route, first we dynamically get the context by using the run_query method for the selected personality and retrieval method. This along with the question will be given to the pipeline created using the selected transformer model to obtain the predicted answer to the question.

By using the official rankings for transformers on the SQuAD dataset [8], we searched for transformer models in Hugging Face’s database that had a very good score on either SQuAD 2.0 or SQuAD 1.1. Using this process, we found three models that had promising scores. For SQuAD 2.0, we found two ALBERT [6] models, one XLARGE and another XXLARGE. For SQuAD 1.1 we found a BERT [2] model. The scores of these models, for their respective SQuAD can be viewed in table 2.

| Model / Score | **BERT** | **ALBERT XLARGE** | **ALBERT XXLARGE** |
|---|---|---|---|
| **F1 Score** | 93.15 | 87.46 | 89.35 |
| **EM Score** | 86.91 | 84.41 | 84.11 |

**Table 2: SQuAD results for used transformer models**

## RESULTS

The solution was evaluated in three different ways. Firstly, we will focus on the accuracy of the answers obtained. After that, we will focus on the retrieval methods and how they compare against each other. Lastly, we will detail the performance of the models based on the dedicated memory used and the time needed for predicting an answer.

### Accuracy of Answers

To compute the accuracy, we created a dataset comprised of 45 questions and answers for three historical figures: Albert Einstein, Adolf Hitler, and Mahatma Gandhi. A few examples are presented in table 3.

The accuracy was computed on multiple combinations of transformers and retrieval methods. To assess the

performance of each model we used the same metrics used for the SQuAD dataset [8], the F1 and EM scores. We started from the SQuAD scoring functions and implemented a small modification for them to be compatible with the output of our test scripts. The F1 measures the similarity between two phrases based on the number of tokens (words) they have in common, while EM is 1 when the predicted answer is identical to the expected answer else is 0.

| Questions | Answers |
|---|---|
| When was Einstein born? | on 14 March 1879 |
| At what subjects was Einstein good in school? | math and physics |
| Why couldn't Hitler apply to the School of Architecture? | he lacked the necessary academic credentials |
| Where was Hitler born? | in Braunau am Inn, a town in Austria-Hungary (in present-day Austria) |
| How many times did Gandhi marry? | four times |
| What did Gandhi study in college? | law and jurisprudence |

**Table 3: Examples of questions and answers in the dataset**

For each of the three transformer models discussed previously, the F1 and EM score was computed with a different context given. The contexts that have been provided to the transformers were the contexts returned by the run_query method using the four retrieval options discussed previously and additionally the whole Wikipedia page of the personality. We need to mention that the results for ALBERT XXLARGE with the context being the whole Wikipedia page was not checked because of its dedicated RAM requirements. All the scores obtained can be viewed in figure 8.

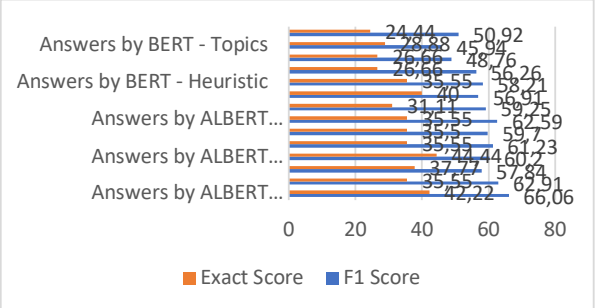



**Figure 8: Accuracy of models depending on the context**

We can see that the best results were obtained using ALBERT XLARGE with the heuristic option, obtaining an F1 of 66.06 and an EM of 42.22. There were other close results, like ALBERT XLARGE and XXLARGE with the paragraph option. We can also see that in general, the BERT model obtained the lowest scores, whereas the highest overall scores were obtained by ALBERT XLARGE. However, we can see that the results obtained by our application are not as good as the results the models obtained on SQuAD. Two factors contribute to this difference in accuracy. Firstly, in this application, we need to dynamically select our context for performance reasons (more about this later). Secondly, because we have a dataset comprised of only 45 questions, we cannot train the transformer on our data to obtain better results, since we would need at least 500 question-answer pairs to be able to properly train the model.

We will now go through the types of answers that the best scoring transformer (ALBERT XLARGE with heuristic option) returned to the questions. Some examples can be seen in table 4.

| | Question | Answer | Predicted Answer |
|---|---|---|---|
| Perfect Matches | Who attacked Gandhi in January 1897? | a mob of white settlers | a mob of white settlers |
| Partial Matches | Where was Gandhi born? | in Porbandar (also known as Sudamapuri), a coastal town on the Kathiawar Peninsula | Porbandar |
| | What is Einstein known for developing? | the theory of relativity | quantum theory, |
| No Matches | Who was Hitler's grandmother? | Maria Anna Schicklgrubber | Geli |
| Detailed Answers | What won him support from many of Germany's most powerful industrialists? | a speech | A speech to the Industry Club in Düsseldorf |

**Table 4: Types of answers returned by the model**

The model managed to return four types of answers. Before examining the answers, it is important to mention that the ALBERT models tend to leave punctuation in the answer, even though they do not respond with a sentence. The first are perfect matches, where the answer provided by the transformer contained exactly the information that was expected. The second are partial matches. This type of answers can appear in multiple forms: they can be either incomplete ("Porbandar" without detailing what and where Porbandar is) or they can contain some of the expected words, but the general answer is different from the expected one ("quantum theory," instead of "the theory of relativity").

The third type of answers are the answers that are completely different from the expected ones. These answers have two causes for appearing. They can either be caused because run_query did not manage to return the target paragraph in the context, or because the transformer was not trained on exactly this type of questions. The last type is answers where the transformer model manages to give a response that is even more detailed than the expected answer ("A speech to the Industry Club in Düsseldorf" instead of "a speech").

**Retrieval Methods**

Since we dynamically select the context that is used by the transformer, we need to examine how many results need to be returned from an index so that the context will contain the target phrase. We constructed a table that contains for every question (out of the 45 questions in the dataset), the index of the topic/paragraph/phase that contained the answers when making a query on Elasticsearch using the processed question.

In general, the target phrase would be returned in the first few results however there were instances where this was not the case. For some question there was a very large number of topics/paragraphs/phrases that needed to be returned so that the context would contain the target phrase. By examining the place where the information requested by those questions appears on the Wikipedia page, we can determine that for Elasticsearch to be able to return the target within only the first few results, the words that appear in the query need to be similar to the words that appear in the indexed document. For the question *"When did Gandhi die?",* on the Wikipedia page the death of Gandhi is described as follows: "*a Hindu nationalist, fired three bullets into his chest from a pistol at close range*". Because the information provided from the words in the question is very different from the words in the Wikipedia source, Elasticsearch can not find these answers easily. The question *"What happened to Gandhi's body after death?"* also does not provide good information because in the Wikipedia article it is mentioned that "*Gandhi was cremated in accordance with Hindu tradition"* completely separately from his death (they are not even in the same topic), because of this Elasticsearch can not infer that this should be in the first results.

We mentioned that we will present how we have chosen the numbers for the heuristic option. We computed the mean for the necessary number of phrases, paragraphs and topics that needed to be returned to obtain the target phrase in the context. There was an average number of 58 phrases, or 10 paragraphs or 3 topics that needed to be returned. However, returning 58 phrases would miss 6 gold phrases, returning 8 paragraphs would miss 7 gold paragraphs and returning 3 topics would miss 11 gold topics. A compromise needed to be made. By comparing the minimum number of results for each question in each index and considering the mean for each index we decided that the heuristic will return 1 topic, 8 paragraphs and 10 phrases. Using this heuristic would mean only 3 questions out of 45 would not get the target phrase in the context.

**Performance**

A way to be able to make sure that the transformer will have the target phrase in the context would be to give a context containing more results. If we would give the transformer all the Wikipedia page for a historical figure, then the context would 100% contain the target phrase. However, doing this impacts the transformer's performance negatively in multiple ways.

Looking at the results from figure 8 we can see that the models that had the whole page as context performed worse than ALBERT XLARGE with the heuristic. This is because when the context becomes too big even if the model is performant, the attention mechanism will not be enough to remember all the long-term dependencies, so the model will fail to answer correctly. Even though ALBERT XLARGE with the heuristic has questions that would be impossible to answer since the context received does not include the target phrase, it still performed better than the models that had the entire page. This means that the transformer model benefits from a context that is smaller in size.

The context is important to the performance of the transformer. If a larger context is given then the model will need more dedicated RAM to be able to predict the answer, also with a larger context, the time needed for the transformer to answer a question is also increased. The dedicated RAM for each scenario can be viewed in table.

| | BERT | ALBERT XLARGE | ALBERT XXLARGE |
|---|---|---|---|
| Article | 5,1 | 5,8 | 7,5 |
| Topics | 2,8 | 2,2 | 3,4 |
| Paragraphs | 2,8 | 2,1 | 3,1 |
| Phrases | 2,8 | 2,1 | 3,1 |
| Heuristic | 2,6 | 2 | 3 |

**Table 5: Mean necessary dedicated RAM for Prediction (GB)**

We can see that giving the whole Wikipedia page to the model can consume as much as 8.3 gigabytes (GB) of dedicated video RAM (this does not appear in table 8 because there only the mean for all three personalities is presented). The models using Elasticsearch to dynamically obtain their context needed a maximum of 3.5 GB RAM, this is an improvement of over 235%. Not only that but the heuristic method is the one that consumes the least amount of RAM, while still obtaining a good result.

Even if we consider that RAM is not the problem, that we have a GPU that has enough memory to easily use the more powerful transformer models on pages that have lots of information, there is still a problem. For a user to be comfortable in using this software the transformer model

needs to be able to answer the questions that are posed in a relatively short time. We can see the average amount of time needed to answer a question for each context variant in table 6.

| | BERT | ALBERT | ALBERT |
|---|---|---|---|
| Article | 14,2 | 33,7 | 79,7 |
| Topics | 1,3 | 3,6 | 6,9 |
| Paragraphs | 1 | 2,7 | 4,9 |
| Phrases | 1,3 | 2,9 | 5,1 |
| Heuristic | 1,2 | 3,8 | 4,9 |

**Table 6: Necessary average time for prediction (in seconds)**

We can see that giving the whole Wikipedia page also has a huge impact on the time necessary for an answer. The models that had all the page given as context performed between 10 and 16 times more slowly than the models which had their context selected dynamically using the run_query method.

## CONCLUSION

### General Conclusion

This paper's objective was to build a conversational agent that can accurately answer questions regarding different historical figures that can be chosen arbitrarily by the users. The results have shown that even if the transformer was not trained of questions for historical figures, it can still perform well for a newly added personality. Considering that the transformers in the SQuAD rankings are tested and trained on a dataset with fixed contexts, this project has shown that even without training on this exact type of questions and with a dynamically chosen context, the transformer model is still able to have a decent performance, a performance superior to that of the related projects.

The entire source code (backend + frontend) that was used in this project is available on GitHub[4]. There you can look through it or download it and run the application in your local environment. You can easily interact with the application on a personality of your choice by following the instructions in the GitHub repository.

### Future Work

This project can be improved in the future. The main directions that need improvement are the way in which the context is determined dynamically using Elasticsearch, the ability to train the transformer models with questions on historical figures and providing sources for the information more reliable than Wikipedia.

The way in which the first direction could be handled is to create a way to derive new meanings from the question. With this we could create a query for Elasticsearch that contains more information about what the user wants to know, enabling Elasticsearch to combat the situations in which the question asked has a completely different structure than the answer that exists in the source document.

For the second direction, we need to create a large enough dataset of questions and answers for personalities so that we could train the transformers on it. This however would require a lot of manpower. To this end, we could let users that want to improve the project submit their own questions and corresponding answers about the historical figures they are interested in. When a large enough dataset is created, we will be able to train the transformer on the dataset, fine-tuning it to better suit the task.

The last direction in veracity. We mentioned in the introduction, that such a QA system would be able to reduce the misinformation in the online environment because a user could easily ask the agent what they want to know, and they would receive the true answer. After the transformer accomplished a high enough accuracy to be reliable, we could add more veridic sources of information like biographical or autobiographical books. We could also implement a system so that along with the answer, we also return the source of that answer. This way a user can easily see where the information originated.

## REFERENCES


1. Bogatu, A., Rotarescu, D., Rebedea, T., & Ruseti, S. (2015). "Conversational Agent that Models a Historical Personality". In *Proceedings of RoCHI 2015 (pp. 81-86).*
2. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
3. E. Haller and T. Rebedea, "Designing a Chat-bot that Simulates an Historical Figure," *2013 19th International Conference on Control Systems and Computer Science*, 2013, pp. 582-589, DOI: 10.1109/CSCS.2013.85.
4. Hirschman, L., & Gaizauskas, R. (2001). "Natural language question answering:the view from here". In: *Natural Language Engineering , Volume 7 , Issue 4 , December 2001*, pp. 275 - 300. DOI: 10.1017/ S1351324901002807
5. Ilie, M., & Rebedea, T. (2017). "Conversational Agents Embodying a Character Using Neural Networks". In *Proceedings of RoCHI 2017* (pp. 125-130).


[4] Github repository for our paper, Accessed August 30, 2021. URL: https://github.com/alex-dima/Conversational-Agent-Embodying-a-Historical-Figure-using-Transformers

6. Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2019). Albert: A lite bert for self-supervised learning of language representations. arXiv preprint arXiv:1909.11942.
7. Internet Growth Statistics. Accessed on June 02, 2021. URL: https://www.internetworldstats.com/emarketing.htm
8. SQuAD Explorer. Accesed June 17, 2021. URL: https://rajpurkar.github.io/SQuAD-explorer/
9. The Future of Truth and Misinformation Online. Accessed June 02, 2021. URL: https://www.pewresearch.org/internet/2017/10/19/the-future-of-truth-and-misinformation-online/
10. Why is History Important and How Can It Benefit Your Future. Accessed on June 03, 2021. URL: https://www.uopeople.edu/blog/why-is-history-important/