# H.A.T. QA Model

Developing a Question-Answering System

**Team 4**

Akshay Govindareddy

Hiranmai Karedla

Tulsi Shrivastava


AIT 526

Introduction to Natural Language Processing

Dr. Kevin Lybarger

May 14, 2023

**Abstract**

The purpose of our project is to create a QA model that allows the user and our algorithm to emulate a query and response dialogue to return the most relevant answer from a text base. This research problem stems from the natural language processing field of information retrieval, in which the system outputs an answer to a user's need for information. Our main source of information is the Stanford Question Answering Dataset (SQuAD), which consists of questions posed by crowdworkers on a set of Wikipedia articles. In this study, we outline how our team employed two approaches (one using traditional NLP methods and a second with deep-learning NLP techniques) and show the results of our analysis. We use Google Co-lab and GMU Hopper to run BERT and Transformers as well as tools like Python, TensorFlow, and Pytorch to train our model. Ultimately, this project resulted in our version of a QA model that emulates the previously conducted SQuAD dataset project. Traditional NLP approaches fall short in retrieving the exact answer for a large set of data, whereas deep learning methods allow for models to be run on large datasets, with better accuracy. Further exploration and deep dive into the methods would allow us to experiment this model on additional, larger datasets.

***Keywords:*** *question-answering, deep-learning, natural language processing*

## I. Introduction

Today, humans have access to a vast sea of information to feed our curiosity and desire for knowledge. So, when asking a question as information receivers, we prefer to get the quickest and most accurate response at our fingertips within seconds, rather than sift through the massive amount of irrelevant data. Derived from the study of information retrieval, Question Answering (QA) NLP models serve as unique solutions because they quickly yield replies in "natural language" as opposed to collecting a list of documents (Ojokoh 2019). The purpose of our project is to create a QA model that allows the user and our algorithm to emulate a query and response dialogue to return the most relevant answer from a text base. Similar to text summarization and similarity scoring, we want to calculate and return the response with the most accuracy. The primary challenge lies in effectively understanding the meaning of the user's query and the model's ability to yield the most accurate and meaningful output. User-generated input text could be layered with ambiguity, lack of structure, or a question that cannot be answered by the text base.

## II. Research Problem

QA models are composed of three main stages that include question analysis, search of documents, and extraction of answers (Ojokoh 2019). In keeping with this structure, there are three key outcomes we hope to achieve: 1) the model should be able to understand the question from a varying user input, 2) answers to the queries should be returned in proper structure and in summarized form, and most importantly, 3) the response should provide the most accurate and correct response to the posed question. If the text does not contain an appropriate response, the model should determine that the question cannot be answered. Our model will be extractive, indicating that the result of the query is certain to exist within a given passage or "context" (typically questions that are for testing basic reading comprehension). According to Jurafsky and Martin, this task is known as open domain QA, where the system is given a factoid question and a large document collection (in our case a passage from Wikipedia article) and must return a span of text extracted from the document (Jurafsky 2000). In contrast, abstractive question answering is when the questions are mostly open-ended and the answers must be generated by the model itself.

If a human were to ask a question based on a text, they would have to read through the entire passage to extract necessary information. Finding certain facts about an object or entity that seldom

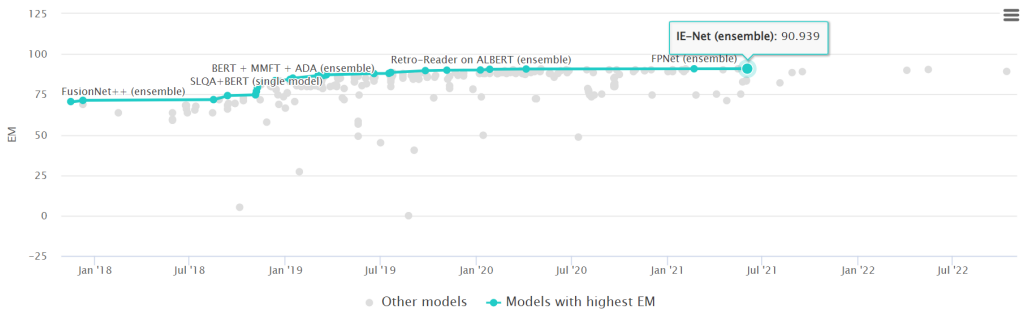
appear for an entire document would be a painstakingly long task. A tool such as the Search/Find on a webpage may be able to yield results, but one may still have to sift through multiple entries to get the desired results. This method is not only time-consuming, but also prone to human error as the result may likely stem from or lead to inaccurate conclusions.

With a question-answering model as our natural language processing solution, the output of the query would be the most relevant and quick answer based on the context given. These systems are derived from earlier rule-based NLP models that were a part of the information retrieval field of study. Traditional NLP techniques were used, and specifications based on question type, answer type, sentence structure, answer length, and more had to be set in place for the model to return a near-correct answer. Newer approaches in information retrieval involve using deep learning and neural networks to train a set amount of data with questions and answers -- and return a higher accuracy of answers. Today, examples of question-answering systems can be found in search engines, FAQ sections, and conversational interfaces.

## III. Related Work

With the rise of information retrieval systems, multiple attempts to create and enhance a QA model have been made. Papers with Code, a website that publishes machine learning research, hosts a leaderboard to track and compare the playing field of datasets and corresponding question-answering models on performance. Popular QA datasets such as WikiQA, TriviaQA, and HotpotQA are available for researchers to use and create their own models using similar techniques. Typically, these QA systems are evaluated based on their EM score (a correct or almost exact match answer) and an F1 Score (measure of the model's accuracy) ("Papers").

Within the Stanford Question Answering Dataset (SQuAD) community itself, there have been dozens of QA model projects conducted. In the original paper, Rajpurkar and authors developed a model with an EM score of 86.8 (indicating a correct or almost exact match answer) and an F1 Score of 89.5 (measure of the model's accuracy) (Rajpurkar 2016). In another Stanford study, authors Park and Lakshman successfully developed an end-to-end deep learning model with high EM and F1 scores. Their main challenge came when hyper-parameter tuning the model and discovering that the SQuAD dataset question-answer pairs may sometimes mark correct answers as incorrect (Park 2017). In a second study published on Analytics Vidhya, author Lavanya also utilized machine learning models to create her model and output answer sentences with above 60% accuracy on her validation sets (S 2021). The Papers with Code screenshot below shows a scatter plot for the SQuAD 2.0 dataset, with models rated on their EM score over time. As of 2021, the IE-Net model is rated at the top, with an EM score of 90.939 and an F1 score of 93.214 ("Papers"). For our project, such a comparison helps to put our model into perspective against the benchmark for the same dataset.



## IV. Methods

Our main source of information is the Stanford Question Answering Dataset (SQuAD), which consists of questions posed by crowdworkers on a set of Wikipedia articles. To run our model, we have the option to select from a variety of 35 topics available to us. In the SQuAD 2.0 dataset specifically, there are more than 108,000 questions on Wikipedia articles available to use. Answers are segments of text in the passage, and the question's words are often synonyms of words in the passage. The file export includes the following fields: topic title, the context (paragraph of information), and relevant question-and-answer pairs shown in the table below (Rajpurkar 2016). The following text box shows the features available to us in JSON file format: id, title, context, question, and answers. A "context" refers to a paragraph or the selection of the passage (the Wikipedia article) in which the answer to a posed question can be found. Given the example of the topic of the element oxygen, the question posed is "How many atoms combine to form oxygen? The answer choices (there can be multiple correct ones) are extracted from the context paragraphs and given as "two atoms" or simply "two".

Dataset({
features: ['**id**', '**title**', '**context**', '**question**', '**answers**'],
num_rows: 87599 })

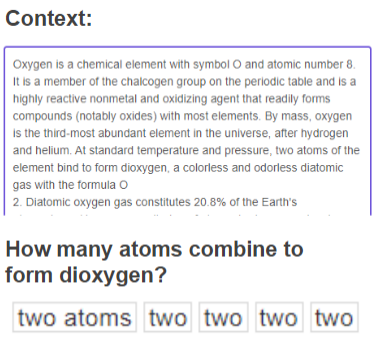
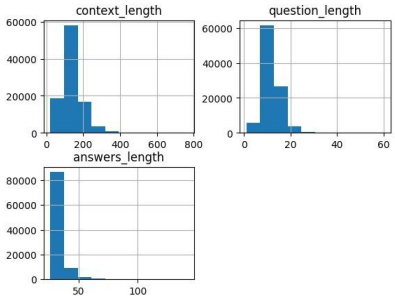



Table 1 and Figure 1 below show summary statistics for the context, question, and answers available to us. The mean context length of 138.3 suggests that the model should be able to process long passages of text. The mean question length of 11.3 indicates that the model should be able to handle relatively short questions, and the mean answer length of 30.1 suggests that the model should be able to generate answers of moderate length. It is important to note that the dataset includes questions with no answers as well, so the QA model may return inconclusive at certain times.

| | context_length | question_length | answers_length |
|---|---|---|---|
| count | 98169.000000 | 98169.000000 | 98169.000000 |
| mean | 138.318777 | 11.303599 | 30.175310 |
| std | 57.709973 | 3.725814 | 7.069772 |
| min | 22.000000 | 1.000000 | 26.000000 |
| 25% | 102.000000 | 9.000000 | 26.000000 |
| 50% | 127.000000 | 11.000000 | 27.000000 |
| 75% | 164.000000 | 13.000000 | 30.000000 |
| max | 766.000000 | 60.000000 | 142.000000 |

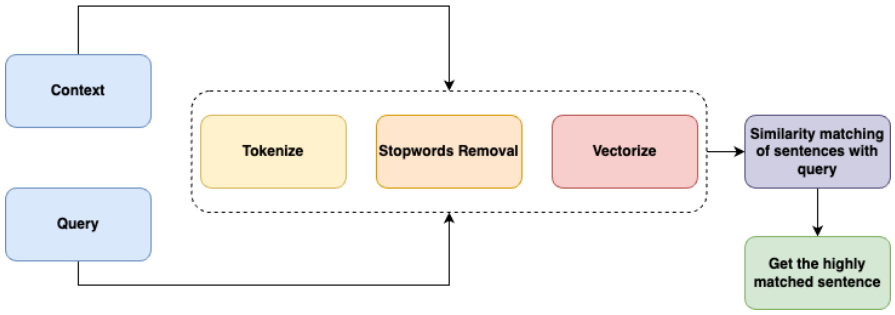



### *Approach and/or Architecture*

Our first approach was to use sentence tokenization and implement a cosine similarity metric to find answers that best match the query. The diagram below outlines our workflow.

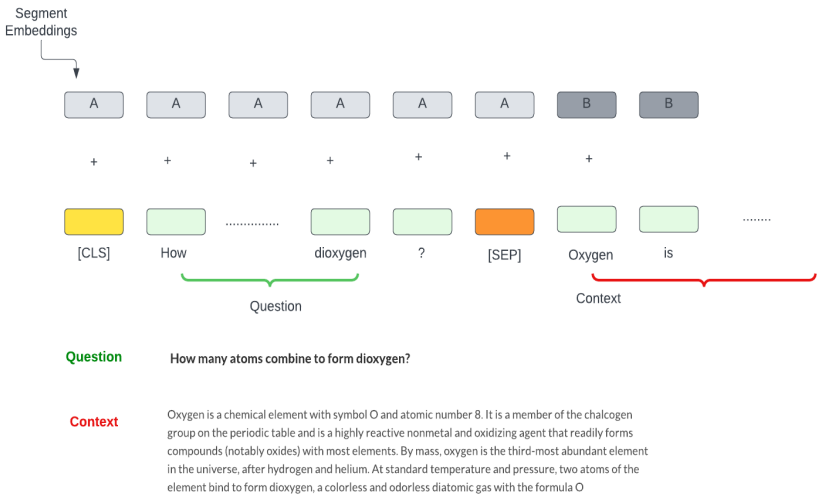
1. Used subset of data - selected the 10000 samples of train data of squad

2. Preprocessing - Tokenizing the context and question, stop word removal
3. Vectorize - Create the vectors for context and question
4. Similarity Matching - calculate similarity matching of context sentence vectors and question vector
5. However, using this method proved difficult in covering all sentence types. We also attempted to map with entities, but this needs a deeper dive in future studies.



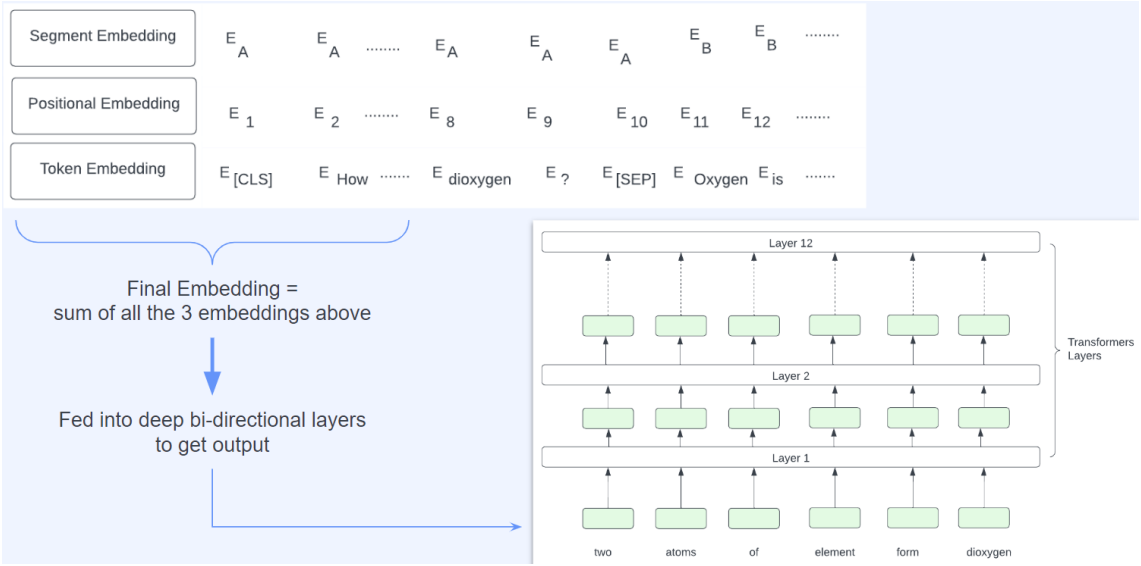

The second approach we want to implement uses deep-learning methods with Transformers and BERT. We began to follow references from the Hugging Face NLP Course to create an extractive BERT-based question answering model ("Question Answering"). The steps are outlined as follows:

1. Import dataset, split into training & validation - Import data from hugging face datasets
2. Convert data into torch data loader - create data loaders for train and validation which torch can understand
3. BERT Input Format - [CLS] question [SEP] context
4. Tokenize the data - use bert tokenizer for better understand of context
5. Preprocessing of Training & Validation Data - Make the preprocessing of samples using stride approach, storing the offset indexes
6. Feed the data and training arguments to Train the model
7. Apply bert-base-cased pre-trained model - Train the model
8. Post processing the predicted span indexes with the probability
9. Map high probability output indices with the context to get the answer
10. evaluation metrics- Exact Match and F1 score are the metrics




For the question-answering system, BERT requires two parameters: the input question and the reference text (context passage) in one concise sequence.

BERT uses token embeddings, which are the sum of the token and segment embeddings, to represent words numerically in a manner that the model can understand. To help the model grasp the context and structure of the input, it uses special tokens ([CLS] and [SEP]) to indicate the beginning and end of the question and the passage, respectively. Segment embeddings, on the other hand, are used to differentiate between sentences in the input. Each token is marked with a segment indicator, either A or B. In the example above, all tokens marked as A belong to the question, while those marked as B belong to the passage. This approach enables the model to understand which part of the input text is the question and which part is the reference text.By using segment embeddings, BERT can differentiate between the question and the reference text, which are two embeddings (for segments "A" and "B") that it learned. The model adds these embeddings to the token embeddings before feeding them into the input layer, allowing it to better understand the relationship between the question and the reference text, ultimately leading to more accurate answers.



The BERT model for question answering (QA) outputs two vectors: a start vector and an end vector. These vectors represent the probabilities of each token being the start or end of the answer span within the context paragraph. The start vector represents the probability that each token is the start of the answer span, while the end vector represents the probability that each token is the end of the answer span. These vectors are computed using a softmax function applied to the output of the final layer of the BERT model. The output of the final layer is a sequence of hidden vectors, one for each input token, which capture the contextualized representations of the tokens in the input paragraph. To obtain the start and end vectors, the softmax function is applied separately to the output for each token, producing a probability distribution over all tokens in the input paragraph. The token with the highest probability in the start vector represents the predicted start position of the answer span, while the token with the highest probability in the end vector represents the predicted end position of the answer span.

Overall, the start and end vectors produced by the BERT model for QA are crucial for identifying the answer span within a given context paragraph. They are computed using a softmax function applied to the output of the final layer of the model, and represent the probability that each token is the start or end of the answer span (Bhageria 2020).

Our team used GMU Hopper to run BERT and Transformers as well as tools like Python, TensorFlow, and possibly Pytorch to train our model. Hopper is equipped with powerful hardware such as 2x Dell PowerEdge R640 and 1x NVIDIA Tesla T4 GPU. NLP techniques employed include text preprocessing, testing the model on training data, fine-tuning the model, text post processing, and determining its performance on the testing dataset. We applied machine learning techniques by fine tuning pre-trained models of BERT and Transformers.

## V. Results

### *System Output*

To provide samples from both traditional and deep-learning methods, we used the topic of the University of Notre Dame. Resulting output is shown in the tables below the context.

In our first example, the context is given as follows, “In 1882, Albert Zahm (John Zahm's brother) built an early wind tunnel used to compare lift to drag of aeronautical models. Around 1899, Professor Jerome Green became the first American to send a wireless message. In 1931, Father Julius Nieuwland performed early work on basic reactions that were used to create neoprene. Study of nuclear physics at the university began with the building of a nuclear accelerator in 1936, and continues now partly through a partnership in the Joint Institute for Nuclear Astrophysics.”

| Example Output 1 | | |
|---|---|---|
| Traditional | Question:<br>'In what year did Jerome Green send his first wireless message?' | |
| | **CORRECT**<br>Our Answer: Around 1899, Professor Jerome Green became the first American to send a wireless message. | |
| Deep Learning | Question:<br>'In what year did Jerome Green | Question:<br>‘Where is that?” |

| | | |
|---|---|---|
| | send his first wireless message?' | |
| | **CORRECT**<br>Our Answer: 1899 | **INCORRECT**<br>Answer: Joint Institute for Nuclear Astrophysics |

From this output, we can arrive at a few conclusions. First, the traditional method managed to produce the correct output - but returned the entire sentence instead of just the span match. While this is satisfactory, it may be the case that the resulting sentence is longer and harder to retrieve in another scenario. Thus, when looking at the deep learning method output, we are satisfied to see that it resulted in the exact year to answer our question. We also noticed that posing a vague question such as “Where is that?” is more likely to return an incorrect answer due to the lack of specificity.

In our second example, the context is given as follows, “The most remarkable episode of violence was the clash between Notre Dame students and the Ku Klux Klan in 1924. Nativism and anti-Catholicism, especially when directed towards immigrants, were cornerstones of the KKK's rhetoric, and Notre Dame was seen as a symbol of the threat posed by the Catholic Church. The Klan decided to have a week-long Klavern in South Bend. Clashes with the student body started on March 17, when students, aware of the anti-Catholic animosity, blocked the Klansmen from descending from their trains in the South Bend station...”

| Example Output 2 | | |
|---|---|---|
| Traditional | Question: 'What type of event did the Klan intend to have at Notre Dame in March of 1924?' | |
| | **CORRECT**<br>SQuAD Answer:<br>a week-long Klavern | **INCORRECT**<br>Our Answer: The most remarkable episode of violence was the clash between Notre Dame students and the Ku Klux Klan in 1924. |

This example outlines a potential shortcoming of the traditional NLP method. The question posed is asked “Which type of event...”, and the model seems to search for an event entity type and ties it back to “the most remarkable episode”. Again, we see that although the system retrieved the highest-scoring sentence, it did not answer the question given in the SQuAD dataset.

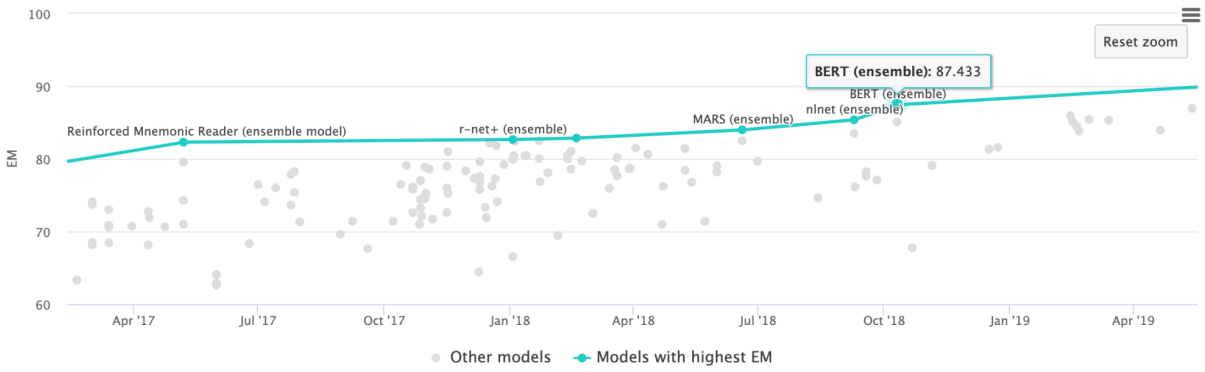
***Result Metrics***

| Size | Exact Match | F1 Statistic |
|---|---|---|
| Train: 25,000 rows<br>Val: 2,000 | 75.75 | 83.58 |
| Train: 50,000 rows<br>Val: 5,000 | 78.92 | 86.91 |
| Whole dataset | 80.21 | 87.13 |

We ran three iterations of our deep-learning model, two with a different subset of the full data and a third attempt with the full dataset. The table below outlines the amount of data used in the training and validation sets as well as the EM scores and F1 statistics. It is shown that the more data we were able to train the model on, the better our performance -- which aligns with machine learning model predictions.

***Results Comparison to Existing Models***

On a range of datasets, the BERT model for question answering has been proved to be successful for QA. There is still space for development, though. In our implementation we were able to achieve an exact match score of 80.21, which is slightly lower than the ensemble BERT model (shown as 87.433 in the Papers with Code chart below) and the single BERT model (85.083) ("Papers"). However, our model is still a significant improvement over the baseline BERT model, and it has the potential to be further improved.




## VI. Conclusions

In summary, the goal of our research project was to create our own version of a question-answering model that could be compared against the existing research and potentially be used in reading comprehension scenarios. In order to fulfill a user's need for information, we desired the most accurate result to the question posed, given a set knowledge base. Our bilateral approach involved initially using basic NLP techniques and then venturing into the deep learning space to achieve greater performance. Results proved that while traditional NLP paths may find similar answers and sentences, they ultimately fall short in retrieving the exact match answer for a large set of data. Deep learning methods allow for models to be run on larger datasets, with better accuracy. Most importantly, structure and preciseness of the wording of the questions makes a big difference in getting accurate results.

The team's greatest obstacle has been loading and running the entire training dataset with the large model packages in CoLab or Jupyter, without any failures. Secondly, understanding the application of Pytorch and Transformers in QA models took some heavy efforts and learning. It is a new thing to learn how deep learning techniques like Bidirectional encodings are useful inorder to perform high level nlp tasks. It is a great chance to utilize almost all the tools taught in the course.

In future iterations of our research, we hope to go further with traditional NLP methods. For deep learning specifically, we hope to test run our model on a larger dataset or perhaps one of the other ones shown in the Papers with Code website (WikiQA, NewsQA). Later, perhaps the model could be used on an entire Wikipedia page to test performance.

## References


Bhageria, D. 2020. “Build a smart question answering system with a fine-tuned Bert.” Medium. Retrieved May 6, 2023. https://medium.com/saarthi-ai/build-a-smart-question-answering-system-with-fine-tuned-bert-b586e4cfa5f5

Jurafsky, D., and James H. Martin. 2000. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition (1st. ed.). Prentice Hall PTR, USA.

“Papers with Code - Squad2.0 Benchmark (Question Answering).” *The Latest in Machine Learning*. Retrieved May 14, 2023 (https://paperswithcode.com/sota/question-answering-on-squad20).

“Question Answering - Hugging Face NLP Course.” Hugging Face. Retrieved April 14, 2023. https://huggingface.co/course/chapter7/7?fw=pt

Rajpurkar, Pranav, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. "Squad: 100,000+ questions for machine comprehension of text." arXiv preprint arXiv:1606.05250.

S, Lavanya. 2022. “End to End Question-Answering System Using NLP and Squad Dataset.” *Analytics Vidhya*. Retrieved April 2, 2023. (https://www.analyticsvidhya.com/blog/2021/11/end-to-end-question-answering-system-using-nlp-and-squad-dataset/).