Ensembling and Data Augmentation for QA

Agam Dwivedi
dwivedi7@wisc.edu

Mohil Patel
mpatel48@wisc.edu

Rahul Chakwate
chakwate@wisc.edu

1 Introduction

Question Answering (QA) in artificial intelligence (AI) focuses on developing algorithms and systems that can answer questions posed in natural language. The field of QA has significant practical applications, including the development of chatbots that can answer customer queries in an automated and conversational manner. ChatGPT is a language model that uses deep learning to generate responses to user input, and QA is a critical component of its functionality. Machine comprehension, another application of QA, involves training machines to read and understand natural language texts and then answer questions about them. Overall, the field of QA has wide-ranging applications and is essential for creating intelligent and conversational AI systems.

Ensembling, oversampling, and question generation have been widely used in machine learning to improve model performance. In our work, we aim to adapt these techniques specifically for the task of Question Answering (QA) using the Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2018). SQuAD is unique in itself, as the answer lies in the context. This allows the model to focus on understanding the context rather than predicting the next word. We also plan to leverage modern Language Models (LLMs) to produce questions and assess their effect on the performance of Question Answering (QA) tasks.

Previous studies have explored the use of ensembling methods, question generation, and other data augmentation techniques in QA tasks. However, none of them has comprehensively analyzed these approaches, specifically on the SQuAD dataset. Our research focuses on evaluating and comparing various data augmentation methods and ensemble models to identify the most effective techniques for enhancing model performance.

The upcoming sections will present previous research related to Question Answering tasks. We will then introduce the SQuAD dataset and discuss the necessary pre-processing steps to prepare it for training. Afterward, we will describe various baseline models and their respective fine-tuning approaches. We will then detail our ensemble technique to enhance model performance, followed by an oversampling approach to address the class imbalance in the dataset. Then, we will cover our Question Generation model, which utilizes LLMs to generate questions. In the final section, we conclude our project and present the individual contributions made by each team member.

2 Background

Question Answering is an extensively studied task in the field of NLP. In this section, we cover some background works in this field. We cover some Question Answering models, using ensemble techniques in QA and Question Generation for question-answering tasks.

BiDAF (Bi-Directional Attention Flow) (Seo et al., 2018) network is an LSTM-based model with the attention mechanism specifically trained for Question Answering. Compared to its predecessors, it uses bi-directional attention, calculated from query-to-context and context-to-query, to improve performance. It also calculates attention for every time step, rather than summarizing the whole query with a single fixed vector, allowing it to capture context better. Attention vectors are passed through the LSTM layer to generate results for the Question Answering task

As the name suggests, BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2019) is a Transformer-based language model. Language model pretraining is shown to be effective for many downstream NLP tasks. Models like GPT-2 (Radford et al., 2019) only use Transformer decoders and pre-trains as unidirectional language models. This left-to-right modeling restricts the representation power of pre-trained pa-

rameters for fine-tuning many downstream tasks. Compared to that, BERT uses Transformer encoders and allows the model to capture bidirectional context. It uses the "Masked Language Model" (MLM) pretraining objective, in which a random 15% of tokens are masked and used for pretraining (unlike left-to-right training in GPT-2). It also uses the "next sentence prediction" task for pretraining, allowing it to get better results in Question Answering and Natural Language Inference tasks.

Ying (2019), in his work, used ensembling to improve performance for QA tasks. In his paper, he used an ensemble of BiDAF+BERT on the SQuAD dataset. The work uses a stochastic ensembling method, where in cases when BERT and BiDAF output disagrees, it probabilistically selects one of them as the correct answer. The results showed an improvement using the ensemble technique.

Now we discuss the BERT variants that we considered for ensemble, specifically RoBERTa, ALBERT and DistilBERT. The BERT language model has areas that could benefit from improvement, and ROBERTa (Liu et al., 2019) is an optimized version developed by Facebook. Modifications made to ROBERTa include training the model for longer periods with larger batches, increasing the amount of training data, removing the next sentence prediction objective, training on longer sequences, and using dynamic masking instead of static masking to create more varied training data. ALBERT (Lan et al., 2020) has architecture is similar to BERT, using encoder layers with GELU activation functions. However, there are three key modifications unique to ALBERT. Firstly, the input-level embedding and hidden-level embedding were separated by factorizing the embedding matrix, resulting in an 80% reduction in parameters with a minor drop in performance. Secondly, cross-layer parameter sharing was introduced to improve efficiency and reduce redundancy, leading to a 70% reduction in the overall number of parameters. Finally, ALBERT used a new loss called SOP (Sentence Order Prediction) to measure inter-sentence coherence, rather than NSP (Next Sentence Prediction) which has the disadvantage of considering both coherence and topic. DistilBERT (Sanh et al., 2020) is an approximate version of BERT that maintains 95% of its performance while using only half of the parameters. It achieves this by eliminating token-type embeddings and keeping only half of the layers in the original BERT model. DistilBERT utilizes distillation, which involves approximating a large neural network (like BERT) with a smaller one. The idea is that the full output distributions of the large network can be approximated using the smaller network once the large network has been trained. This process is similar to posterior approximation, with Kulback Leiber divergence being a key optimization function used for this purpose.

Question Generation is used to generate a synthetic dataset for the QA task. Alberti et al. (2019) improve on generating synthetic question-answering corpora by using a novel method to generate (C, Q, A) (context, question, and answer pairs). In their approach, they train three separate models and use roundtrip consistency to select high-quality pairs. For passage C, first, they train a model to extract an answer A (step 1). Using the generated A with passage C, in step 2, they output the question Q using a separately trained model. And lastly, in step 3, they generate a new answer A', using a question-answering model. Step 4 helps decide whether to output the (C, Q, A) pair. Only those pairs that output the same A and A' are allowed. This check helps ensure high quality in output pairs. Overall they show that pretraining with a synthetic dataset generated from this method leads to better results in QA tasks on SQuAD2.0 (Rajpurkar et al., 2018) and NQ datasets (Kwiatkowski et al., 2019).

Unlike the previous approach, Lopez et al. (2021) tried to answer Question Generation with only one model. Many Question Generation models use multiple models and additional features like answer awareness (Alberti et al., 2019) and extra-linguistic features to generate questions. Lopez et al. (2021) treat question generation as a language modeling task in their approach. They used pretrained GPT-2 (Radford et al., 2019) and fine-tuned the model with the SQuAD dataset (Rajpurkar et al., 2018) as input data for language modeling. Each training example has a context paragraph and associated question(s) with a delimiter in between. Training this way allows the model to generate questions similar to language modeling tasks. They tried different delimiters and input formats and showed that the questions generated were on par with state-of-the-art models.

3 Experimental Setup

Our objective is to ask questions about a given document and identify the relevant answers as text

spans within the document. We will be using pre-trained language models for our experiments, which are not specifically trained for question answering. These models will be fine-tuned on the SQuAD dataset as a first step.

3.1 The SQuAD Dataset

The SQuAD dataset (Rajpurkar et al., 2018), also known as the Stanford Question Answering Dataset, is widely used as a benchmark for retrieval-based question answering in academic settings. The dataset includes questions posed by crowd-workers based on Wikipedia articles, along with the relevant context where the answers can be found. Our models are designed to predict the specific start and end points of the answer within the context. The dataset comprises 87,599 training samples and 10,570 validation samples, and includes features such as example ID, question, context, and a set of ground-truth answers. During training, each question has only one answer, but during evaluation, a list of answers is provided due to the possibility of multiple correct answers for some ambiguous questions within a given context. The following example shows how for a given question and the context, there are multiple possible answers.

Question: *"Where did Super Bowl 50 take place?"*

Context: *"Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California. As this was the 50th Super Bowl, the league emphasized the golden anniversary with various gold-themed initiatives, as well as temporarily suspending the tradition of naming each Super Bowl game with Roman numerals (under which the game would have been known as Super Bowl L), so that the logo could prominently feature the Arabic numerals 50."*

Answers: *"Santa Clara, California", "Levi's Stadium", "Levi's Stadium in the San Francisco Bay Area at Santa Clara, California."*

3.2 Pre-processing the dataset

To make our input data understandable for the models, we need to convert natural text into numerical representations. To achieve this, we tokenize the input question and context, where each word corresponds to one token. The models' objective is to predict the starting and ending tokens of the answer. Each token has two labels, a start label, and an end label, and the models' task is to predict these labels. As shown in Figure 1, the context is split into tokens, and the starting and ending tokens of the answer are marked with 1, while all other tokens are marked with 0.

To accommodate the fixed tensor length that the models can handle, we have limited the context length to a maximum of 384. For longer contexts, we divide them into multiple features, and to avoid any incorrect splits in the middle of an answer, we overlap the features. We then convert the ground-truth answer locations to tokens and generate labels that we can use to fine-tune our model.

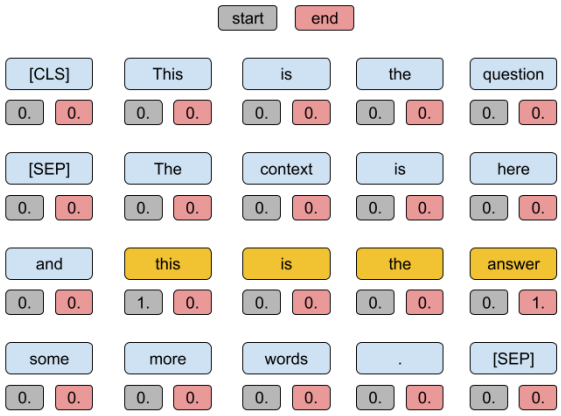

Figure 1: Label Creation Process

3.3 Fine-tuning the models

The models that we are using in our experiments are pretrained language models and not specifically designed for the Question Answering task. To make them work for our problem, we fine-tuned them on the SQuAD dataset after pre-processing it as described earlier. We obtained the pretrained base language models using the Hugging Face API (Wolf et al., 2020) and fine-tuned them for 3 epochs using PyTorch, which took around 2-3 hours of training per model on Google Colab TPU. The models that we fine-tuned include BERT-base, RoBERTa-base, ALBERT-base, and DistilBERT-base. We have made our trained models available

on the Hugging Face hub, so they can be downloaded and used by anyone without the need for retraining from scratch, which saves computational resources.

3.4 Post-processing and Evaluation

Our model produces two raw logits for each token node in the feature vector - one for the start and the other for the end of the answer. We need to post-process these raw outputs to get the actual start and end indices of the answer from the context. To obtain probabilities, we applied softmax to the input logits. We then filtered out invalid outputs, which are [start,end] pairs where the start index is greater than or equal to the end index. We considered only the top n best logits from each list, where n is 20 in our case, based on the answer length and context size parameters. Assuming independence between the events "The answer starts at the start index" and "The answer ends at the end index", the probability that the answer starts at the start index and ends at the end index is calculated as follows.

$$start_{prob}[index_{start}] * end_{prob}[index_{end}]$$

We calculated the probability of all valid pairs and selected the pair with the highest probability as the model's predicted answer with its corresponding confidence score. Then, we mapped the start and end indices of the answer with the text from the context. To evaluate our model on the test dataset containing about 10,000 samples, we compared the predicted answers with the ground-truth answers and computed the percentage of exact matches and the F1 score.

4 Ensemble Method

Ensembling techniques have historically proved to be effective in improving model performance. We believe such techniques can also be adapted to improve the performance of individual QA models, which we explore in our work.

4.1 Approach

We used a post-processing script to obtain a confidence score for each predicted answer, which was then used to create an ensemble. For each question-context pair, we chose the predictions of the model with the highest confidence score among the models considered. We based our ensemble technique on the idea that the model would predict correct answers with high confidence but would be less confident for wrong answers. This is because in the case of wrong answers, the model can be confused between multiple answers and will predict them with low probability.

4.2 Results

The results of the ensembling technique discussed in the previous section are presented in Table 1. The first ensemble (Ensemble 1) is created by combining the base models of BERT, ALBERT, and RoBERTa that were fine-tuned on the Squad dataset. It was observed that Ensemble 1 doesn't perform any better than individual models in terms of Exact Match and F1 score. Since this ensemble did not improve the performance over RoBERTa-base, the decision was made to replace it with a lightweight model, DistilBERT, whose individual performance was lower than the other two. The second ensemble (Ensemble 2) outperformed all three individual models in terms of F1 score by 0.7% and in Exact Match by a significant 2.2%.

Next, we tried to combine different variants of the same model, and Ensemble 3 was created by combining the base, xlarge, and xxlarge models of ALBERT. However, this ensemble did not improve the performance over individual models, possibly due to the weakness of ALBERT-base. Thus, we removed ALBERT-base in Ensemble 4, which outperformed the two individual models by approximately 1.4% on Exact Match and 0.6% on F1. We then combined the ALBERT-xlarge and ALBERT-xxlarge models with RoBERTa models in Ensembles 5 and 6, respectively. Both ensembles showed improvements in performance, especially Ensemble 6, which had the biggest improvement of approximately 2% on Exact Match and 1% on F1.

Based on our experiments and the results in Table 1, we drew the following conclusions:

- We initially believed that an ensemble of multiple language models would always perform better than individual models in that ensemble.
- We thought that taking the maximum confidence score among individual models would result in at least the same level of performance as the best model in the ensemble.
- However, the experiment results showed that this is not always true. For example, in Ensemble 1 and Ensemble 3, an individual model out-

Model	Exact Match (%)	F1 Score (%)
BERT-base	81.15	88.58
ALBERT-base	81.94	89.70
RoBERTa-base	**85.20**	**91.78**
Ensemble 1	84.73	90.75
BERT-base	81.15	88.58
ALBERT-base	81.94	89.70
DistilBERT-base	79.57	86.92
Ensemble 2	**84.12**	**90.41**
ALBERT-base	81.94	89.70
ALBERT-xlarge	87.40	93.58
ALBERT-xxlarge	**87.82**	**94.14**
Ensemble 3	87.67	93.48
ALBERT-xlarge	87.40	93.58
ALBERT-xxlarge	87.82	94.14
Ensemble 4	**89.20**	**94.77**
ALBERT-xlarge	87.40	93.58
ALBERT-xxlarge	87.82	94.14
RoBERTa-base	85.20	91.78
Ensemble 5	**89.44**	**94.79**
ALBERT-xlarge	87.40	93.58
ALBERT-xxlarge	87.82	94.14
RoBERTa-large	87.41	93.74
Ensemble 6	**89.74**	**95.10**

Table 1: Results of ensemble performance. Each ensemble is a combination of the 2 or 3 models mentioned above it. Exact Match: Percentage of predictions that exactly match with true answers. F1 Score: F1 metric of the predicted v/s true answers.

performed the ensemble's performance. This contradicts our belief that ensemble will always improve the performance.

- However, in other ensembles (2, 4, 5, and 6), the ensemble models outperformed the individual models contributing to that ensemble.
- Therefore, we can conclude that the ensemble outperforming individual models in many of the cases but not necessarily in all of them.
- We cannot make any conclusive generalization using our current ensembling approach. However, there may be other ensembling techniques that could lead to better improvements, such as weighted averaging, using MLPs to predict the weights, etc., which we plan to investigate in the future.

5 Oversampling

In machine learning, oversampling is a technique used to address imbalanced datasets, where one class of data is significantly underrepresented compared to the others. This imbalance can lead to biased models that generalize poorly. Oversampling involves creating synthetic examples of the minority class by duplicating or generating new instances. The goal of oversampling is to balance data distribution across different classes, leading to more accurate and representative models. It is a widely used technique in machine learning and has been shown to improve model performance.

5.1 Approach

Past works have shown the importance of oversampling in Question Answering tasks. To understand the dataset characteristics better, we first did some preliminary analysis and plotted the imbalance in the SQuAD dataset. As shown in Figure 2, the dataset has a very high percentage of 'what' type questions compared to the rest of the question types.

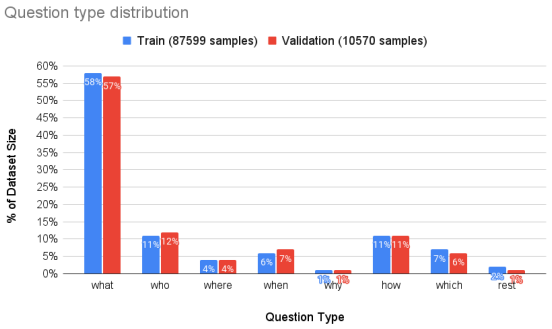

Figure 2: Distribution of Question Types in SQuAD

Based on this understanding of the dataset characteristics, we designed the approach to oversample the underrepresented question types by varying the ratio of the max question type to the min question type. The hope is that oversampling will reduce the trained model's bias and improve performance.

5.2 Results

Figure 3 shows the results that we got from oversampling. As we can see, increasing the balancing ratio does not significantly increase the score. And as we got to higher oversampling, we saw a decrease in the performance.

To understand why we did not see the improvement we hoped for, we plotted the question type

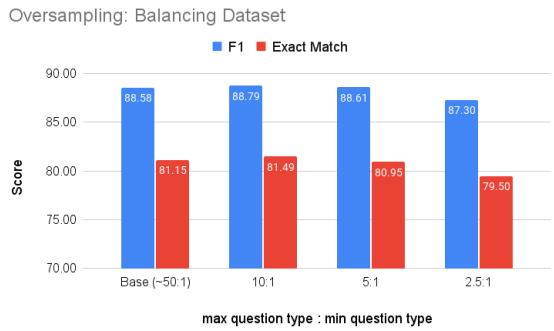

Figure 3: Oversampling underrepresented question types

distribution for incorrect predictions for the base model without any oversampling. As shown in Figure 3, the distribution of wrong predictions is almost the same as the train and validation split. This similarity implies that each question type gets the same percentage of incorrect questions. Thus, the base model, even without any oversampling, is learning each question type equally well. Therefore adding extra samples for underrepresented question types is not helping much.

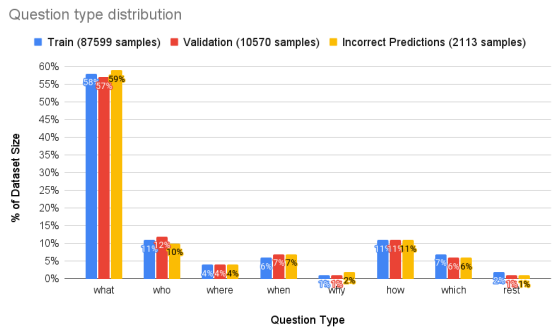

Figure 4: Question type distribution for base model's incorrect predictions

6 Question Generation

Question Generation is a critical task for NLP. In question generation, we train a model to take text corpus as input and generate questions as the output. Some models only require text corpus input, whereas others require both text corpus and answer strings to generate questions. It is used in chatbots, customer service applications, etc., to generate questions that can help further refine the queries. It is also used in educational systems to create practice questions.

6.1 Approach

Past works in question generation have used question generation to augment the dataset with newer data points and generate better results. In our approach, we wanted to see how effective LLMs can be for question generation. LLMs can be used for many tasks, including question generation. We used a T5-based model (Raffel et al., 2020) fine-tuned for question generation. This model takes context and answers as input to generate questions. We generated new questions using the same context and answer from the SQuAD dataset.

Input (C)	... in 1903, boston participated in the first modern world series, going up against the pittsburgh pirates ...
(1) $C \rightarrow A$	1903
(2) $C, A \rightarrow Q$	when did the red sox first go to the world series
(3) $C, Q \rightarrow A'$	1903

Table 2: Example of how synthetic question-answer pairs are generated.

In addition to using the same context and answer to generate new questions, we also tried to train a model to extract new answer strings from the context. As shown in Table 2, we extract new answers, which are passed through the T5-based question generation model to generate new questions. We believed that implementing this approach would enhance the quality of the questions generated. However, we did not observe satisfactory outcomes. We believe that modeling start and end tokens independently during prediction leads to substandard answer phrase quality.

6.2 Results

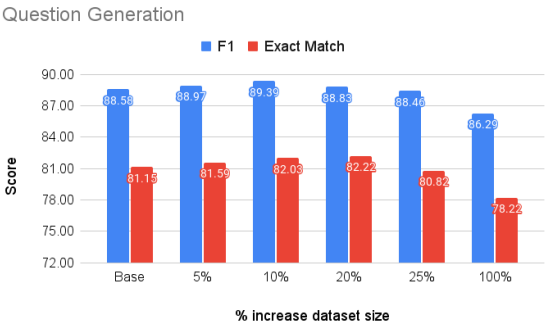

Figure 5: Question Generation using T5-based model

Figure 5 shows the results by augmenting the dataset with newly generated questions. We see the performance increasing as we increase the dataset size by 10%. After that, we start seeing a decreasing trend. This result shows that we can use LLMs to improve performance for question-answering tasks by augmenting datasets with new questions.

7 Conclusion & Future Works

This work shows that ensembling techniques and question generation can help improve performance for question-answering tasks, specifically for the SQuAD dataset. On the other hand, oversampling did not show improvement because the base model (base-bert) itself can learn each question type equally well. The ensembling approach of using the model confidence as the proxy to decide which model to believe in has shown improvements and matches the intuition of the confidence parameter. As for question generation, we can use LLMs to augment the dataset with new questions and improve performance. In this study, we only showed the results for T5. But as many LLMs are available now, we can repeat this study for them and do a comparative study.

For question generation models, many of them require context and answer as input. This requirement of having an answer as an input string limits the quality of new questions that we can create. During our experiments, we tried to train a model which takes the context as input and generates an answer string as the output. This BERT-based model took context as input and generated start and end positions in the context as output, which points to the answer string. Although we could not get good results here, further study showed that joint modeling of start and end tokens could lead to better results. This is one potential direction that can be explored. We also think that input features like Part-of-Speech tagging and syntactic parsing can improve the quality of generated answer string. And thus, it will enhance the quality of generated questions too.

8 Acknowledgements

We thank Google for their free Colab TPUs which allowed us to train and fine-tune our models in a short time. We also thank Prof. Junjie Hu for their valuable inputs throughout the project.

9 Contributions

- **Rahul Chakwate:** Implemented the ensemble module by creating the post-processing script to obtain a confidence score for each predicted answer. Also, setup the Question Generation model for data augmentation.
- **Mohil Patel:** Performing data augmentation using oversampling methods for question types which are under-represented in the dataset. Trained and fine-tuned the Question Generation model.
- **Agam Dwivedi:** Trained the context to answer model and utilized it to generate new responses for the Question Generation task. Trained and fine-tuned the Question Generation model.

References

Chris Alberti, Daniel Andor, Emily Pitler, Jacob Devlin, and Michael Collins. 2019. Synthetic qa corpora generation with roundtrip consistency.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

Luis Enrico Lopez, Diane Kathryn Cruz, Jan Christian Blaise Cruz, and Charibeth Cheng. 2021. Simplifying paragraph-level question generation via transformer language models.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2018. Bidirectional attention flow for machine comprehension.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Huggingface's transformers: State-of-the-art natural language processing.

Andrew Ying. 2019. Really paying attention : A bert + bidaf ensemble model for question-answering.