# Robust Question Answering: A study on Self-Attention Mechanism and Data Augmentation

Stanford CS224N Default Project: Robust QA track
TA mentor: Kendrick Shen; External collaborator: No; Sharing Project: No

**Shicheng Xu**
Department of Computer Science
Stanford University (SCPD)
lukexu@stanford.edu

**Yechao Zhang**
Department of Computer Science
Stanford University (SCPD)
yechaoz@stanford.edu

## Abstract

This project aims to build a robust question answering (QA) system that can adapt to unseen domains with only a few training samples from the domain. Our system focuses on the **model limitation** of 512 tokens in full self-attention mechanism, and the **data limitation** of only having 127 questions per out-of-domain training set. For the model limitation, we implement `DistilBertLongForQuestionAnswering`, which increases the sequence length by modifying the self-attention layer in DistilBERT to use Longformer self-attention. For the data limitation, we experiment with various data augmentation techniques to increase the out-of-domain dataset size. Without any ensemble or additional pretraining, our best single model achieves EM/F1 scores of **42.661/60.185** on the test set.

## 1 Introduction

Training deep learning models from scratch typically requires enormous accelerator resource and large-scale datasets. The trend in the last few years has been pretraining models on abundantly-available unlabeled text data with a self-supervised task, then finetuning on downstream tasks with less computational power and data[1]. Recent advents of huge pretraining models, such as GPT-3[2] and T5[3], have shown effectiveness in boosting many NLP tasks.

Instead of chasing the largest SOTA pretrained model, in this project, we are allowed to only use DistilBERT[4]. DistilBERT adopts the same network structure as BERT, but reduces the number of attention layers from 12 to 6 by applying knowledge distilation. One main limitation is that its full self-attention mechanism has computational and memory[1] requirements that are quadratic to the input sequence length[6]. The pretrained DistilBERT model has set its max sequence length to 512. This might hurt the QA performance badly on long context sequences.

In this project, the in-domain (ID) and out-of-domain (OOD) datasets are analyzed to confirm the existence of sequence length problem in all datasets. We identify that the baseline model only sets max length to 384, and it could be extended to 512 to make full use of the pretrained DistilBERT model without any model change. We further convert the pretrained DistilBERT model to use the Longformer[7] self-attention for processing longer sequences. Experiments are conducted on sequence lengths of 384, 512, and 1024 to show the effectiveness of increasing sequence length.

Although modern Transformer-based pretrained language models have proven to be effective at extractive QA, they struggle to generalize well to domain-agnostic settings. A common remedy is to perform finetuning to help the model learn the domain-specific distribution. However, this remains as a challenge when the domain-specific data is underrepresented, which can cause the system to underperform in the few-shot setting during test time. In this project, we need to build a robust QA system under the few-shot setting where the out-of-domain dataset are in only .254% of an in-domain datasets. To tackle the challenge, we perform data augmentation techniques at word, character, and sentence level to extend the out-of-domain training sets and study their impact on model performance.

[1] Rabe et al.[5] presented an self-attention implementation that only requires $O(\sqrt{n})$ memory.

## 2 Related Work

Reducing the quadratic complexity of transformer self-attention mechanism has been a popular research topic in the deep learning community, see the survey by Tay et al.[8] for a summary of related works until 2020. Notebly, Sparse Transformer[9], Longformer[7], and BigBird[10] are all sparse attention methods aimed at solving the problem. Sparse Transformer introduces strided attention that only applies on some ($\text{query}_i$, $\text{key}_j$) pairs. Longformer presents sliding window attention and global attention where some global tokens could attend to all tokens. BigBird adds random attention on top of Longformer. Transformer-XL[11] is orthogonal to sparse attention and relies on segment-based recurrence to connect two adjacent blocks. More recently, the Perceiver model family ([12], [13], [14]) introduces latent self-attention and uses a Transformer-style cross-attention module to map inputs to a smaller number of latents so that the size of latent self-attention is independent of input size.

We study Longformer in this project for three reasons. First, Longformer is the foundation of Poolingformer[15]. Poolingformer is ranked as top performance[2] in the Google's Natural Questions challenge[16]. Second, Longformer self-attention could be applied to any Transformer-based model. This allows us to reuse training weights from the pretrained DistilBERT model. Third, making question tokens as global tokens that attend to all tokens matches the intuition of QA task.

Data augmentation (DA) refers to strategies for increasing the amount of training examples from the existing data. It has shown promising results for boosting model performance on various NLP tasks[17]. Wei et al.[18] show that simple word level augmentation such as synonym replacement and random deletion can boost performance and reduce overfitting for small training sets.

## 3 Approach

In this section, we first deep dive into the baseline model and report our findings on its limitations. We then present our main approaches: `DistilBertLongForQuestionAnswering` in section 3.2 to tackle the model limitation, and data augmentation in section 3.3 to tackle the data limitation.

### 3.1 Baseline

The provided baseline in the project handout[19] loads the default uncased pretrained DistilBERT tokenizer and model, and trains on only in-domain datasets. The `DistilBertForQuestionAnswering` model adds a single linear layer on the DistilBERT model as the classification head for QA.

In the tokenzier, the (question, context) pair is splitted into chunks with a fixed size. Each chunk is labeled and sent to the model separately. If the answer is in the span of the context, then the chunk is labeled with true start index, otherwise the answer will be marked with the [CLS] token index. With this labeling strategy, the model may implicitly learn from out-of-span chunks that the answer is not in the span, but it would not have the chance to learn the long dependency across multiple chunks.

We find the actual tokenizer output is different from what the example describes in the project handout. In HuggingFace tokenizer class[3], the stride parameter means the number of overlapping tokens between the end of $\text{chunk}_i$ and the start of $\text{chunk}_{i+1}$. The last chunk will be padded to max sequence length with zero tokens (0). Formally, the correct representation of the second chunk in the handout example should be $c_2 = [\text{CLS}]q[\text{SEP}]p^2[\text{SEP}]0\cdots0$ with $p^2 = \{p_{244}, p_{255}, \cdots, p_{500}\}$, and 113 zero tokens are appended to $c_2$ until its length is 384. The key takeaway is that the max sequence length in the baseline model is 384, which is smaller than the default max length 512 in DistilBERT. To our best knowledge, we are the first team that identifies this limitation in the baseline model.

### 3.2 `DistilBertLongForQuestionAnswering`

In the full self-attention of the original Transformer, every token can attend to other tokens, so the computation complexity scales quadratically to the input sequence length. Longformer[7] presents two sparse attention mechanisms to address this challenge: Sliding Window and Global Attention. The Sliding Window attention employs a fixed-size window attention surrounding each token. Given a fixed window size $w$, every token only attends to $w$-nearby tokens. However, this will prevent question tokens from attending to context tokens that are outside of its sliding window. The Global

[2] `https://ai.google.com/research/NaturalQuestions/leaderboard`

[3] `https://huggingface.co/docs/transformers/main_classes/tokenizer#transformers.PreTrainedTokenizerFast`

Attention solves this problem by treating question tokens as global tokens that attend to all tokens across the sequence. The computation complexity of Longformer self-attention is linear to the sequence length (Table 1).

| | Full Self-Attention | Chunking | Longformer Self-Attention |
|---|---|---|---|
| Computation Complexity | $O(n^2)$ | $O(m \times w^2)$ | $O(n \times (w + q))$ |

Table 1: Computation complexity comparison between different attention mechanisms. $n$: sequence length, $w$: window size or chunk size, $m$: number of chunks, $q$: average question length.

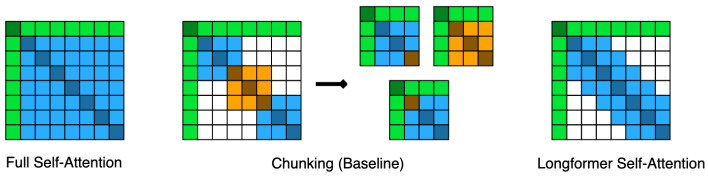



Figure 1: Comparison between full self-attention, chunking, and Longformer self-attention. Green: question tokens; Blue, Orange: context tokens in different chunks.

There are two main differences between sliding window in Longformer and chunking in the baseline model (Figure 1). First, attention weights in Longformer are jointly learned across sliding windows, but they are separately learned in difference chunks in the baseline. Second, the true label is visible to all sliding windows in Longformer, whereas for chunking, it is only useful in some chunks that contain the answer. Moreover, the global attention allows the question to attend to all context tokens.

The Longformer self-attention layer could replace the self-attention layer of any Transformer model. The original Longformer model is converted from a RoBERTa checkpoint. We follow the Colab[4] by the Longformer author to convert the DistilBERT checkpoint to use Longformer self-attention. The implementation style of the self-attention layer in DistilBERT is different from BERT or RoBERTa, we implement `DistilBertLongSelfAttention`, map the attention weight matrices from DistilBERT to LongFormer, and add a linear layer for the attention outputs. The original position embeddings only have 512 positions, we adopt the same approach in Longformer to extend the embedding size to 2048 by repeating position embeddings.

Next, we implement `DistilBertLongForQuestionAnswering` to apply the converted checkpoint on our QA task. In Longformer input attention mask, 0 means no attention, 1 means local attention, 2 means global attention. A DistilBERT sequence has the format of [CLS]$q$[SEP]$p$[SEP], which is different from a RoBERTa sequence [CLS]$q$[SEP][SEP]$p$[SEP]. We set global attention mask before the first separation token. Before feeding the attention mask into `DistilBertLongSelfAttention`, we also need to convert it so that -10000.0 means no attention, 0 means local attention and 10000.0 means global attention for computing local and global attentions. There is no existing implementation we can directly reuse and we independently solve all technical issues to get the correct implementation.

### 3.3 Data Augmentation

In this project, we explore augmenters at three levels: word, character, and sentence. Within each level, we apply data augmentation techniques to fully understand their impact on model performance (see Figure 2). To reduce the inherent randomness in DA, for each element of an out-of-domain train sets triple (context, question, answer), we apply only one technique from the most befitting augmenter for the element. Figure 3 details our augmentation process. For the word and character augmenters, there are two parameters: strength and number of augmentations $n_{\text{aug}}$. For the sentence augmenter, only the number of augmentation is needed. Define $p_{\text{word}}$ as the strength parameter that governs the percent of the words changed for a given sequence and $p_{\text{char}}$[5] as the strength parameter that governs the percent of characters changed per token. We allow these strength parameters to vary for all techniques within each augmenter. We also allow $n_{\text{aug}}$ to vary for each augmenter. We finetune the baseline model using the augmented out-of-domain train sets to search for the optimal parameters.

[4] `https://colab.research.google.com/github/allenai/longformer/blob/master/scripts/convert_model_to_long.ipynb`

[5] By definition, $p_{\text{char}}$ only applies to character augmenter.

| Technique | Definition | Example |
|---|---|---|
| no operation | | … Zoom and [shutter speed] For action or crowd shots, a fast [shutter speed] is a key factor. "... wildlife or people in action on the street--faces change within …" said Arnold,"a fast [shutter speed] is helpful in shooting the several hundred photos you might need to get …" |
| delete | Randomly remove words (not stop words) that are not in the answer span or its recurrences in the paragraph | … Zoom and [shutter speed] For action or crowd shots, a fast [shutter speed] is a factor. "... wildlife or people in action on the ~~street~~ -- faces change within …" said Arnold, " a fast [shutter speed] is helpful in shooting the several ~~hundred~~ photos you might need to get …" |
| swap | Randomly swap two words (not stop words) that are not in the answer span or its recurrences in the paragraph | … Zoom and [shutter speed] For action or crowd shots, a fast [shutter speed] is a key factor. "... wildlife or people in action on the street -- faces change within …" said Arnold, " a [shutter speed] is helpful in shooting the several hundred photos you might **to need** get …" |
| substitute | Randomly a word (not stop words) that are not in the answer span or its recurrences in the paragraph and replace the word with its synonym using wordnet | … Zoom and [shutter speed] For action or crowd shots, a fast [shutter speed] is a key factor. "… wildlife or **citizenry** in action on the street -- **confront** change within …" said Arnold, " a [shutter speed] is helpful in shooting the several hundred photos you might need to get …" |

(a) Word Augmenters. Apply to context paragraphs only. Yellow = Answer Span. Green = other Answer Span recurrences in the context. We ensure that no changes are made to these spans.

| Technique | Definition | Example |
|---|---|---|
| no operation | | 'shutter speed' |
| insert | Randomly select a word in the answer text and inject new characters randomly. | '**shuptter** speed' |
| delete | Randomly select a word in the answer text and remove characters randomly. | '**shuttr** speed' |
| swap | Randomly select a word in the answer text and randomly swap adjacent characters. | 'shutter **pseed**' |
| substitute | Randomly select a word in the answer text, and some original characters will be replaced with new characters randomly. | 'shutter **hneed**' |

(b) Character Augmenters. Apply to answers only. We ensure that the same operation is applied to the answer span and its recurrences in the context paragraph.

| Technique | Definition | Example |
|---|---|---|
| no operation | | What will contribute to a satisfactory photo of a running lion in the wild? |
| back translation | Apply back translation to the question using Google Translate with different languages | English → French → English:<br>What will contribute to a satisfactory photo of a racing lion in nature? |

(c) Sentence Augmenter. Applies to questions only.

Figure 2: Definition of augmenters. For all augmenters where contexts or answers are modified, we also change the `answer_start` index to match the modification.

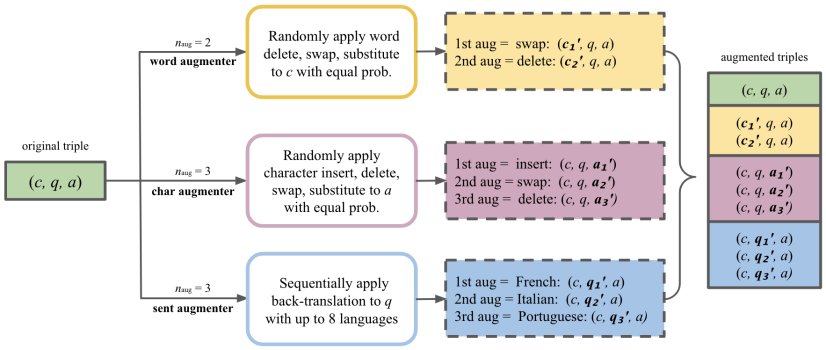



Figure 3: Augmentation process. Given an OOD train set triple $(c, q, a)$, each augmenter only applies to one element of the triple while keeping the other two unchanged. As noted in Figure 2, word augmenters apply to $c$ only, character augmenters apply to $a$ only, and sentence augmenters apply to $q$ only. The augmentation process is governed by the optimal parameters. For example, if $n_{\text{aug}}$ for the word augmenter is 2, then we apply word augmentation to $c$ twice, for which each is done by randomly performing only one of the three techniques. If the first operation is swap and the second operation is delete, then we have $c \rightarrow c'_1$ and $c \rightarrow c'_2$ ($q$ and $a$ remain unchanged), resulting two new triples $(c'_1, q, a)$ and $(c'_2, q, a)$. The example process here results in 9 triples including the original.

# 4 Experiments

This section focuses on our unique findings, instead of repeating what is already explained in the project handout. We will present our findings from data analysis, immediate improvements over baseline, and experiment details on `DistilBertLongForQuestionAnswer` and data augmentation.

## 4.1 Data

In this project, we are provided with three in-domain (ID) reading comprehension datasets (Natural Questions, NewsQA and SQuAD) and three different out-of-domain (OOD) datasets (RelationExtraction, DuoRC, RACE). 50000 examples are provided for each ID training set, whereas only 127 examples are provided in each OOD training set for finetuning.

We apply the pretrained DistilBERT tokenizer on both ID and OOD training dataset, and compute a histogram of number of chunks based on different max sequence lengths and stride. At default max length 384 and stride 128, only the Natural Questions dataset has long (question, paragraph) pairs with more than four chunks. Other datasets have shorter (question, paragraph) pairs, especially for OOD data, see Figure 4. When we increase max length to 512, the number of chunks drops significantly, for example, dataset RACE no longer have (question, paragraph) pairs with three or more chunks. Increasing max length to 1024 and stride to 512 further reduces the number of chunks of most datasets to less than two, except for Natural Questions.

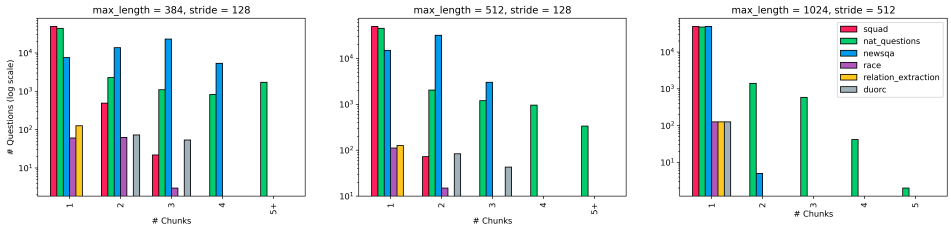



Figure 4: Number of chunks at different max length and stride length.

## 4.2 Evaluation method

We report Exact Match (EM) score and F1 score on OOD validation sets for all experiments and on OOD test sets for selected models.

## 4.3 Experimental details

Unless otherwise specified, the experiments were performed using a batch size of 16 and a learning rate of 3e-5. Training experiments on ID dataset or combined datasets were trained through 3 epochs. Finetuning and data augmentation experiments on OOD datasets were trained through 10 epochs.

### 4.3.1 Immediate improvements over baseline

The baseline model was only trained on ID datasets and set max length to 384. We conducted following experiments with simple changes to improve the model performance.

**Experiment 0: Finetune OOD.** We finetuned the baseline with OOD training sets and achieved EM/F1 scores of 32.46/48.49 on OOD validation sets.

**Experiment 1: Cotrain ID and OOD.** We jointly trained the ID and OOD datasets following many previous year's reports (e.g. [20]), then repeat experiment 0.

**Experiment 2: Increase max length to 512.** We modified the tokenizer to set max length to 512 and keep stride length as 128, then repeated experiment 0 and 1. For ablation study, we also finetuned at max length 384. Training runs on combined datasets in this experiment was trained through 5 epochs.

### 4.3.2 `DistilBertLongForQuestionAnswering`

We took findings from immediate improvements and experimented with our DistilBERTLong model.

**Experiment 3: Finetune OOD with different max lengths.** We used a converted model from the best model from Experiment 2 and finetuned on OOD data with max length at 1024, 1536, and 2048.

**Experiment 4: Cotrain ID and OOD with max length 1024.** From previous experiments we know that cotraining ID and OOD produces more robust model performance. For this experiment, we used a converted model from the default uncased pretrained DistilBERT checkpoint. Due to resource constraint and observation from experiment 3, we cotrained ID and OOD with max length 1024, and stride length at 128 and 512. We set batch size to 8 and learning rate to 1.5e-5.

#### 4.3.3 Data Augmentation

**Experiment 5: Word augmenter for contexts.** For the context paragraph, we decided to use the word-level augmenter. Word level augmenter is more suitable for context since character level techniques could generate too many nonexistent words that confuse the model and sentence level techniques such as back translation could undeliberately change the contextual semantics. Here, we explored three word level DA techniques: delete, swap, substitute using the nlpaug library[6]. Nlpaug is an versatile and easy-to-use library that is designed to automatically generate more synthetic data with various types of augmenters to improve model performance.

We first searched the optimal strength parameter $p_{\text{word}}$ using a set of 10 values $\{0.05, 0.1, 0.2, \ldots, 0.9\}$ for each of the 3 techniques by finetuning the baseline with the augmented OOD training sets while fixing $n_{\text{aug}}$ = 2. We selected the optimal $p_{\text{word}}$ for each technique based on the average of the OOD validation sets F1 and EM scores (Figure 6a). Using the optimal $p_{\text{word}}$ parameter for each technique, we then finetuned on $n_{\text{aug}}$ = $\{1, 2, 4, 8, 16\}$ to select the optimal $n_{\text{aug}}$ (Figure 6b). In total, we conducted $10 \times 3 + 5 = 35$ experiments.

**Experiment 6: Character augmenter for answers.** Character augmenter is more suitable for answers since we want to preserve the answer length. In addition, these nonexistent but look-alike words in the augmented answers (and the answer span recurrences in the context) could force the model to learn the surrounding context to better extract the answer span.

Character augmenter has two strength parameters $(p_{\text{word}}, p_{\text{char}})$. To keep the grid searching manageable, we restricted $p_{\text{word}}$ to 3 values $\{0.05, 0.20, 0.40\}$, which produced the highest average of F1 and EM scores across all three techniques in the word augmenter (Figure 6a). For $p_{\text{char}}$, we searched from 6 values $\{0.05, 0.1, 0.2, ..., 0.5\}$, where the 0.5 cap is selected to avoid a word deviating too much from its original appearance. Following the same search routine for the word augmenters, we selected the optimal $(p_{\text{word}}, p_{\text{char}})$ for each of the four techniques while fixing $n_{\text{aug}} = 2$ (Figure 7a). We then used the optimal strength parameters and search for the optimal $n_{\text{aug}}$ from the same 5 values $\{1, 2, 4, 8, 16\}$ (Figure 7b). In total, we conducted $3 \times 6 \times 4 + 5 = 77$ experiments.

**Experiment 7: Sentence augmenter for questions.** We applied back translation to questions using gooletrans[7], which is a Python library that implements the Google Translate API. This is a simple technique that can better preserve the semantic equivalence of the questions compared to other augmenters. We conducted 8 experiments under $n_{\text{aug}}$ = $\{1, 2, \cdots, 8\}$ with 8 languages[8] (Figure 8).

**Experiment 8: Combine all augmenters.** We combined word, character, and sentence augmenters with best parameters from Experiment 5, 6, and 7, then conducted finetuning experiment (Figure 3).

### 4.4 Results

**Results 1: Immediate improvements over baseline.** We find that combining both ID and OOD training set is a simple but effective way to train a robust model. Simply increasing max length to 512 improves the EM score over baseline by 11.95%. Combining both tricks, we achieve EM/F1 scores 36.91/51.17 on validation sets, and EM/F1 scores 41.307/58.875 on test sets.

**Results 2: DistilBERTLongForQuestionAnswering.** Training Longformer takes more memory and longer training time. In Experiment 3, we only run OOD finetuning with different max lengths, and find that max length 1024 performs the best. With max sequence length at 1024, we need to reduce the batch size to 8 to avoid OOM. Our best single model cotrains ID and OOD, sets max sequence length to 1024 and stride length to 512, achieves EM/F1 scores 38.48/53.70 on validation sets, and EM/F1 scores 42.661/60.185 on test sets.

**Results 3: Data Augmentation.** Table 2 summarizes the optimal parameters for each augmenter obtained so far by performing the finetuning routine explained in 4.3.3. With these optimal parameters, we observe performance boost with DA for each augmenter and the combined over the Baseline and

[6] `https://github.com/makcedward/nlpaug`

[7] `https://pypi.org/project/googletrans/`

[8] The 8 languages are French, Italian, Portuguese, German, Dutch, Norwegian, Spanish, and Irish.

Baseline+finetune models (Table 3. Our combined DA achieves EM/F1 scores 35.08/49.39 on the validation sets, and EM/F1 scores 40.275/57.185 on test sets.

| Augmenter | $n_{\text{aug}}$ | $(p_{\text{word}},\ p_{\text{char}})$ | | | |
|---|---|---|---|---|---|
| | | insert | delete | swap | substitute |
| word | 1 | $(\times,\ \times)$ | $(0.40,\ \times)$ | $(0.20,\ \times)$ | $(0.05,\ \times)$ |
| character | 1 | $(0.05,\ 0.10)$ | $(0.20,\ 0.20)$ | $(0.20,\ 0.40)$ | $(0.05,\ 0.10)$ |
| sentence | 6 | $(\times,\ \times)$ | $(\times,\ \times)$ | $(\times,\ \times)$ | $(\times,\ \times)$ |

Table 2: Optimal number of augmentations parameter $n_{\text{aug}}$ and strength parameters $(p_{\text{word}},\ p_{\text{char}})$ for each augmenter. Symbol $\times$ indicates the parameter is not applicable to the augmenter.

| ID | Methods | F1 | EM | Performance Gain (F1) | Performance Gain (EM) |
|---|---|---|---|---|---|
| 0 | Baseline (ID) | 47.72 | 30.63 | 0 | 0 |
| 0 | Baseline (ID)+Finetune(OOD) | **48.49** | **32.46** | 1.61% | 5.97% |
| 1 | Cotrain (ID+OOD) | **51.53** | **35.86** | 7.98% | 17.07% |
| 1 | Cotrain (ID+OOD)+Finetune(OOD) | 50.81 | 34.82 | 6.48% | 13.68% |
| 2 | len=512 (ID) | 50.67 | 34.29 | 6.18% | 11.95% |
| 2 | len=512 (ID)+len=512 (OOD) | 50.17 | 34.29 | 5.13% | 11.95% |
| 2 | len=512 (ID+OOD) | **51.17** | **36.91** | 7.23% | 20.50% |
| 2 | len=512 (ID+OOD)+len=384 (OOD) | 50.27 | 35.86 | 5.34% | 17.07% |
| 2 | len=512 (ID+OOD)+len=512 (OOD) | 50.96 | 36.65 | 6.79% | 19.65% |
| 3 | len=1024 (OOD) | **47.56** | **34.03** | - | 11.10% |
| 3 | len=1536 (OOD) | 47.41 | 32.72 | - | 6.82% |
| 3 | len=2048 (OOD) | 47.41 | 32.98 | - | 7.67% |
| 4 | len=1024, stride=128 (ID+OOD) | 50.07 | 35.08 | 4.92% | 14.53% |
| 4 | len=1024, stride=512 (ID+OOD) | **53.70** | **38.48** | 12.5% | 25.63% |
| 5 | DA (OOD) - Word Augmenters | 48.57 | 33.51 | 1.78% | 9.40% |
| 6 | DA (OOD) - Character Augmenters | 49.24 | 31.94 | 3.19% | 4.28% |
| 7 | DA (OOD) - Sentence Augmenters | 49.10 | 34.82 | 2.89% | 13.68% |
| 8 | DA (OOD) - Combined Augmenters | **49.39** | **35.08** | 3.50% | 14.53% |

Table 3: Validation Results for Experiments

## 5 Analysis

**Attention Mechanism.** Our experiments clearly show that as the max sequence length increases, model performance improves significantly. To our surprise, the stride length makes a big difference for sequence length 1024. Increasing stride length basically increases the number of chunks that the true answer can appear. We think this helps the model to learn more about how to find answer in different contexts. Due to resource constraint and our observation in Experiment 3, we do not further increase the sequence length to run more experiments. Here we report the training speed and memory usage at different sequence lengths in Table 4[9].

| Sequence length (Model) | Batch size | Train speed (it/s) | Train time | Memory (MB) |
|---|---|---|---|---|
| len=384 (DistilBERT) | 16 | 82 | 50min | 6639 |
| len=512 (DistilBERT) | 16 | 60 | 1h | 8997 |
| len=1024 (DistilBERTLong) | 8 | 13 | 3h10min | 11781 |
| len=2048 (DistilBERTLong) | 4 | 6.4 | 6h30min | 12035 |

Table 4: Training speed and memory comparison at different sequence length.

**Data Augmentation.** Despite its performance boost on the overall OOD datasets, DA could hurt model performance for long sequence context. Table 5 in Appendix breaks down the augmenters' EM/F1 scores by each OOD dataset. For short contexts (RE), we observe steady EM/F1 improvements across all augmenters. For long contexts (Race, Duorc), however, EM/F1 scores fail to improve except for the character augmenter. For sentence augmenters, simply paraphrasing the question does not improve for the long-sequence context QA. Word-level operations on context are inherently noisy so that the resulting paragraph might no longer be semantically equivalent even though stop words are applied. One limitation is that our parameter searching is performed separately, so they might not

[9]Train time is estimated on ID datasets for 1 epoch.

work optimally jointly. Character-level operations on answers are able to improve EM/F1 for Race and Duorc. This confirms our hypothesis that these nonexistent but look-alike words in the augmented answers and the answer span recurrences in the context help the model learn the surrounding context.

**Prediction Example.** In Figure 5, we show predictions from different models on an example in which the context has 677 words. We can see that the model with sequence length 512 is not able to capture the long dependency, while our `DistilBERTLongForQuestionAnswering` model predicts the correct answer. Applying the character augmenter helps the model extract the correct answer span from a very long-sequence context that bears complex relations among multiple subject nouns. The other two augmenters are not able to do so.

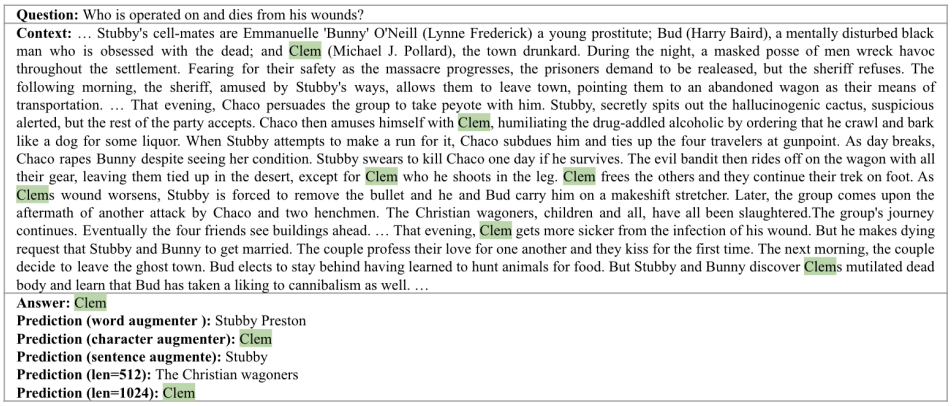



Figure 5: Example case from TensorBoard. The green-highlighted text are answer span and its recurrence in the context paragraph.

## 6 Conclusion

In this project, we studied self-attention mechanisms and data augmentation to tackle model limitation and data limitation in the RobustQA course project. Our key takeaways are as follows:

We confirmed that given limited data size, cotraining with other larger datasets in the same task is a simple but effective way to improve the model performance. We found that further finetuning does not always improve the model performance, the deterioration is likely due to overfitting in OOD data.

Increasing the sequence length in self-attention mechanism should be prioritized given available accelerator resource. As Kaplan et al.[21] suggested in the scaling laws, the computation budget should be spent primarily on larger models, without dramatic increases in training time or dataset size. Our study showed that even with a small model like DistilBERT, increasing attention size and making use of Longformer self-attention can improve the model performance significantly, without the necessity of increasing the running time quadratically. By increasing the sequence length to 1024 and stride length to 512, our best single model achieved EM/F1 scores of **42.661/60.185** on the test set.

Lastly, DA is effective when having a tight budget on accelerator resource. Here, we proposed a simple DA pipeline that employs three types of augmenters. We performed extensive ablation studies on the parameters for each augmenter to carefully analyze the impact on performance gain. We found that performing character-level DA on answer spans is particularly effective for long-sequence contexts. Our final DA pipeline achieved EM/F1 scores of **40.275/57.185** on the test set.

There are further improvements on Longformer to try, such as using pooling layer as described in Poolingformer. Our approach is orthogonal to other methods that have been explored by CS224N students such as multi-task and ensemble. And there are many exciting ongoing research works that we can follow to solve long sequence problems beyond question answering. We hope to take the learning and apply it in real-world problems.

## References

[1] Exploring transfer learning with t5: the text-to-text transfer transformer. `https://ai.googleblog.com/2020/02/exploring-transfer-learning-with-t5.html`. Accessed: 2022-03-11.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[3] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[4] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[5] Markus N Rabe and Charles Staats. Self-attention does not need $o(n^2)$ memory. *arXiv preprint arXiv:2112.05682*, 2021.

[6] Constructing transformers for longer sequences with sparse attention methods. `https://ai.googleblog.com/2021/03/constructing-transformers-for-longer.html`. Accessed: 2022-03-11.

[7] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[8] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*, 2020.

[9] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[10] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33:17283–17297, 2020.

[11] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[12] Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning*, pages 4651–4664. PMLR, 2021.

[13] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021.

[14] Curtis Hawthorne, Andrew Jaegle, Cătălina Cangea, Sebastian Borgeaud, Charlie Nash, Mateusz Malinowski, Sander Dieleman, Oriol Vinyals, Matthew Botvinick, Ian Simon, et al. General-purpose, long-context autoregressive modeling with perceiver ar. *arXiv preprint arXiv:2202.07765*, 2022.

[15] Hang Zhang, Yeyun Gong, Yelong Shen, Weisheng Li, Jiancheng Lv, Nan Duan, and Weizhu Chen. Poolingformer: Long document modeling with pooling attention. In *International Conference on Machine Learning*, pages 12437–12446. PMLR, 2021.

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

[17] Steven Y. Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, Teruko Mitamura, and Eduard Hovy. A survey of data augmentation approaches for nlp. 2021.

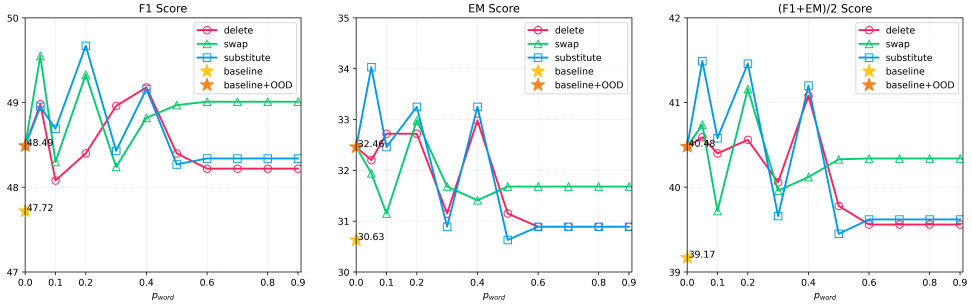
[18] Jason Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. 2019.

[19] Stanford CS 224N Staff. Cs 224n default final project: Building a qa system (robust qa track). 2022.

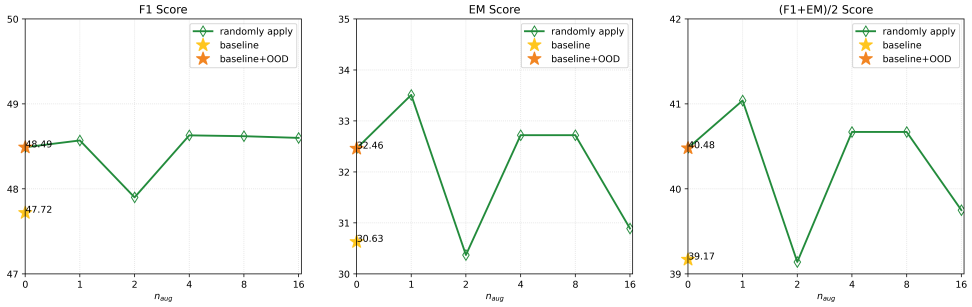
[20] Siyun Li, Xi Yan, and Yige Liu. Dam-net: Robust qa system with data augmentation and multitask learning. 2021.

[21] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

# A Appendix




(a) Grid search for the optimal $p_{\text{word}}$ while keeping $n_{\text{aug}} = 2$. Based on the average F1 and EM scores, the optimal $p_{\text{word}}$ = 0.40, 0.20, 0.05 for delete, swap, substitute respectively.



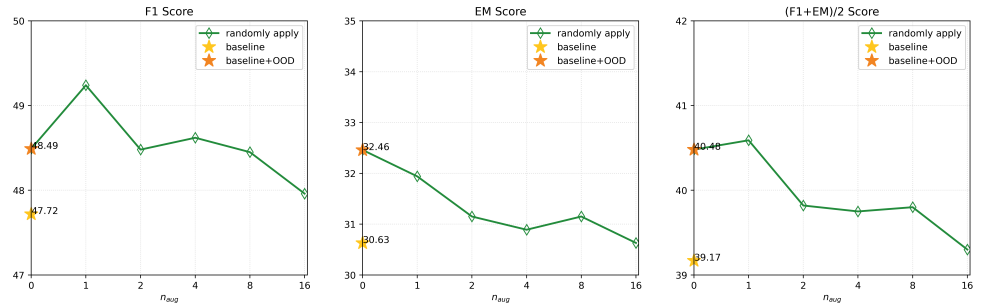

(b) Grid search for the optimal $n_{\text{aug}}$ using the optimal $p_{\text{word}}$ from (a). The resulting optimal $n_{\text{aug}}$ is 1. We then record the F1 and EM obtained using the optimal $p_{\text{word}}$ and $n_{\text{aug}}$ in Table 3 for word augmenters.

Figure 6: Grid search optimal parameters for word augmenters.

| Augmenter | Race | | Duorc | | RE | | All | |
|---|---|---|---|---|---|---|---|---|
| | F1 | EM | F1 | EM | F1 | EM | F1 | EM |
| none (Baseline (ID)) | 36.65 | 21.88 | 39.53 | 30.16 | 66.84 | 39.84 | 47.72 | 30.63 |
| none (Baseline (ID)+OOD) | 31.99 | 17.97 | 40.62 | 28.57 | 72.73 | 50.78 | 48.49 | 32.46 |
| word | 32.50 | 19.53 | 39.14 | 27.78 | 73.94 | 53.12 | 48.57 | 33.51 |
| character | 39.27 | 24.22 | 41.52 | 31.75 | 66.83 | 39.84 | 49.24 | 31.94 |
| sentence | 33.42 | 20.31 | 38.38 | 27.78 | 75.34 | 56.25 | 49.10 | 34.82 |
| combined | 33.08 | 19.53 | 38.87 | 30.16 | 76.06 | 55.47 | 49.39 | 35.08 |

Table 5: OOD F1 and EM validation scores by dataset.


(a) Grid search for the optimal $(p_{\text{word}}, p_{\text{char}})$ while keeping $n_{\text{aug}} = 2$. Based on the average F1 and EM scores, the optimal $p_{\text{word}}$ = 0.05, 0.20, 0.20, 0.05 for insert, delete, swap, substitute respectively. The optimal $p_{\text{char}}$ = 0.10, 0.20, 0.40, 0.10 for insert, delete, swap, substitute respectively.




(b) Grid search for the optimal $n_{\text{aug}}$ using the optimal $(p_{\text{word}}, p_{\text{char}})$ from (a). The resulting optimal $n_{\text{aug}}$ is 1. We then record the F1 and EM obtained using the optimal $(p_{\text{word}}, p_{\text{char}})$ and $n_{\text{aug}}$ in Table 3 for character augmenters.

Figure 7: Grid search optimal parameters for character augmenters.

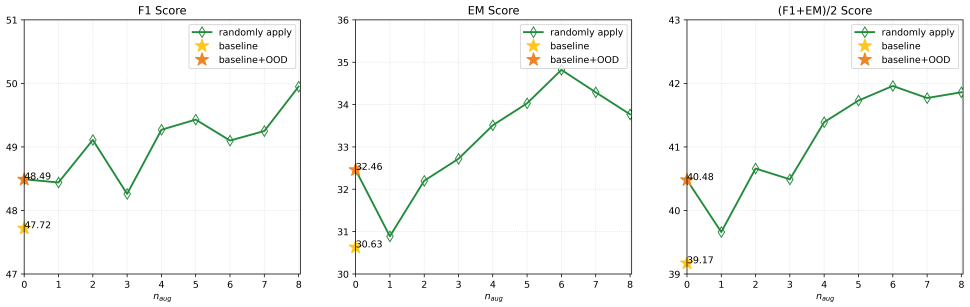



Figure 8: Grid search for the optimal $n_{\text{aug}}$ for sentence augmenter (backtranslation). The optimal parameter is selected based on the average of F1 and EM scores. The resulting optimal $n_{\text{aug}}$ is 6. We then record the F1 and EM obtained using the optimal $n_{\text{aug}}$ in Table 3 for sentence augmenters.