NLP report

Kirollos Luka _ 52-4363
Youssef Maged _ 52-13803

Methodology

In this project, we aimed to develop a model capable of answering questions based on a given context, utilizing the Stanford Question Answering Dataset (SQuAD). Our approach was guided by three main research questions, which shaped the design and evaluation of our model. These are discussed in detail below.

---

1. What Architecture to Use?

The first key decision was selecting the appropriate neural architecture for our model. We considered several options, including Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs), and Bidirectional LSTMs (Bi-LSTMs). Below is a comparison of these architectures:

Architecture	Description	Advantages	Disadvantages
RNN	Processes input sequentially, maintaining a hidden state	Simple and computationally less intensive	Suffers from vanishing gradient problem; poor long-term dependency handling
LSTM	A type of RNN with memory cells and gates to manage information flow	Handles long-range dependencies well; mitigates vanishing gradients	More computationally expensive than RNNs

Bi-LSTM	Extends LSTMs by processing the input in both forward and backward directions	Captures context from both past and future tokens	Doubles computational cost; longer training time

After evaluating the trade-offs, we selected **Bi-LSTM and LSTMs** as our architecture due to its superior performance in understanding contextual relationships, which is particularly beneficial for question answering tasks.

---

2. What Type of Encoding to Use?

The second major decision involved the method of text encoding. We evaluated two main strategies: character-level encoding and word-level encoding. The following table summarizes the strengths and limitations of each:

Encoding Type	**Advantages**	**Disadvantages**
Character-Level Encoding	More flexible with unseen words and misspellings; useful for morphologically rich languages	Requires longer sequences and more training time; less semantic information per token
Word-Level Encoding	Faster training and richer semantic representation; aligns well with pretrained embeddings	Cannot handle out-of-vocabulary words; less robust to spelling errors

To further explore the impact of encoding strategies, we implemented and trained two separate models: one utilizing **character-level encoding** and the other employing **word-level encoding**. This practical comparison allowed us to observe the theoretical advantages and disadvantages of each approach in real-world performance. Character-level encoding demonstrated robustness to out-of-vocabulary tokens, while word-level encoding achieved better semantic understanding and faster convergence.

---

3. What Hyperparameters to Use?

Finally, we explored various hyperparameters that could influence our model's performance. The parameters we focused on included:

These hyperparameters were fine-tuned through experimentation, and their final values were selected based on validation accuracy and model generalization.

---

4. Now we will explore the two models we created

1) Character level LSTM model.

Preprocessing Techniques

Preprocessing is essential in preparing text data for NLP tasks like question answering. It involves techniques such as getting unique contexts, padding, and tokenization.

1. Getting Unique Contexts

In NLP tasks, words can appear in different contexts with different meanings. To help the model understand these variations, we group sentences or questions with similar intents and remove redundancies.This ensures the model gets exposure to unique examples and diverse contexts.

2. Padding

Padding is used to make all input sequences of equal length by adding padding tokens to shorter sequences. This is important for batch processing, as neural networks require consistent input lengths.

3. Tokenization

Tokenization splits text into smaller units called tokens (words, subwords, or characters). We used Character level.

To determine the most effective hyperparameters for our model, we ran a comprehensive grid search by iterating through **36 different combinations** of optimizer types, batch sizes, dense unit configurations, and learning rates. The following hyperparameters emerged as the best-performing combination based on validation accuracy, training stability, and overall model generalization:

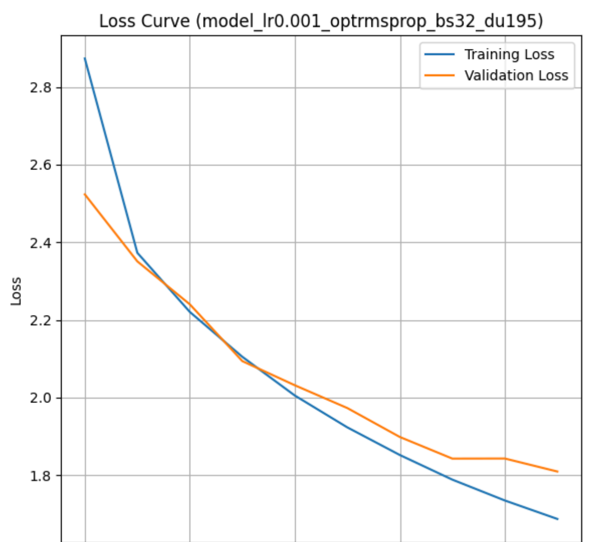
- **Optimizer**: We experimented with several optimization algorithms, including **RMSprop** and **Adam**. After evaluating convergence behavior and performance, we selected **Adam** due to its adaptive learning rate and consistent results across multiple runs.

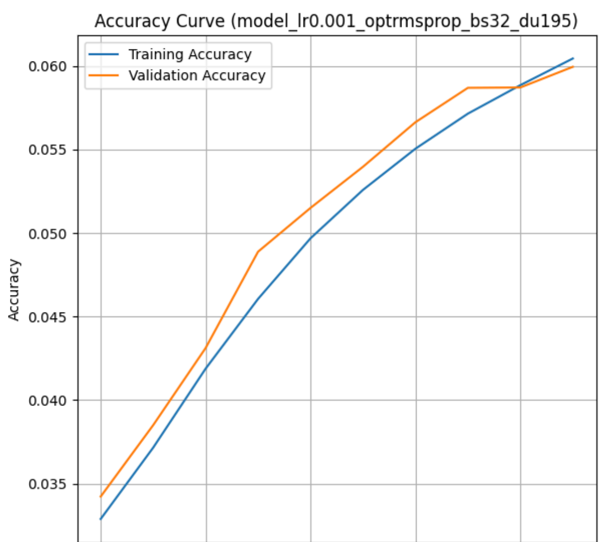
- **Batch Size**: We tested batch sizes of **16**, **32**, and **64**. A batch size of **32** provided the best balance between convergence speed and training stability.

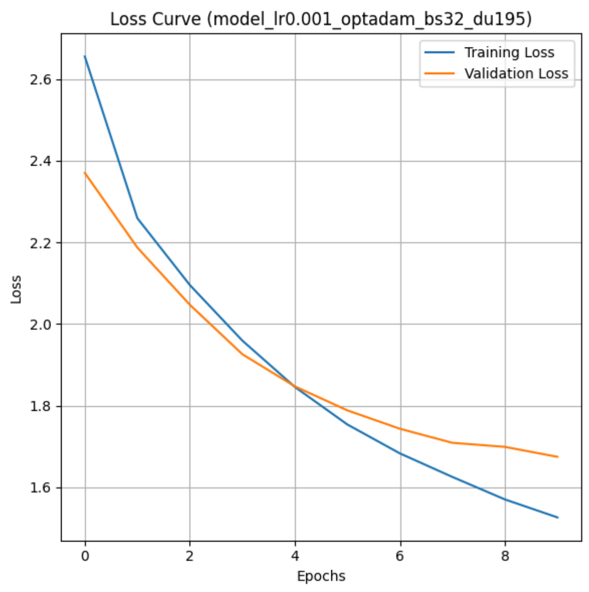
- **Number of Dense Units**: We compared configurations with a dense layer size equal to the **vocabulary size** and another with **vocab size / 2**. The model using dense units equal to the vocabulary size consistently achieved better accuracy and generalization.

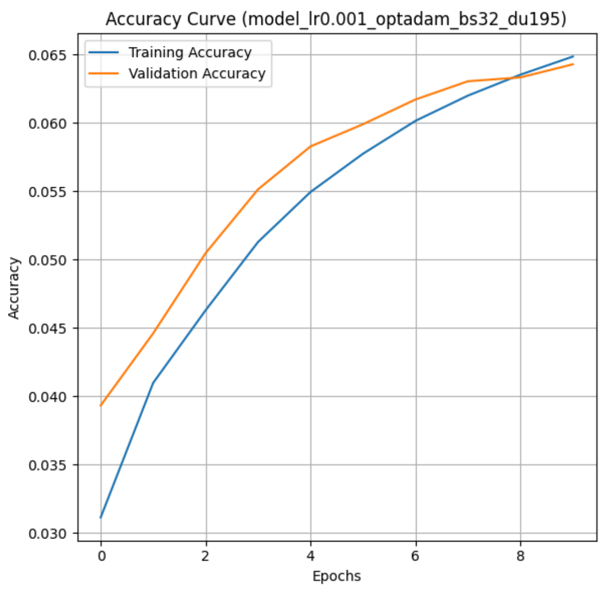
- **Learning Rate**: Two learning rates were evaluated: **0.001** and **0.0001**. A learning rate of **0.001** demonstrated faster convergence without sacrificing stability, making it the optimal choice.

Here are some of the promising results we achieved.









More on the results can be seen here:

https://drive.google.com/file/d/1_sgVAlrZ4Vtv9TCC4lUZIhY-i2CTShsC/view?usp=sharing

---

Findings

The model was trained over 80 epochs using an LSTM architecture with cuDNN acceleration enabled and dropout regularization applied to reduce overfitting. Throughout the training process, several important trends were observed in the accuracy and loss metrics for both training and validation sets.

Accuracy and Loss Trends

- Training Accuracy steadily increased from 2.4% in epoch 1 to a maximum of 7.6% by epoch 45, indicating that the model was able to learn patterns from the training data.
- Validation Accuracy, however, plateaued early, reaching a maximum of 4.8% in epoch 14, before gradually decreasing and stabilizing around 4.0–4.3% in later epochs.
- Training Loss consistently improved, dropping from 2.62 in epoch 1 to 0.46 by epoch 80, showing that the model minimized error on the training set.

- In contrast, Validation Loss increased steadily from 2.31 in epoch 1 to over 3.08 by epoch 80, suggesting limited generalization ability despite the use of dropout.

Score Summary (Selected Epochs)

Epoch	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
1	2.4%	4.2%	2.62	2.31
10	5.6%	4.7%	1.95	2.46
14	6.2%	4.8%	1.74	2.50
45	7.6%	4.1%	0.97	2.87
80	7.5%	4.3%	0.46	3.08

Interpretation

Although dropout was employed as a regularization technique to combat overfitting, the model still exhibited a widening gap between training and validation performance. This suggests that the dropout technique alone was not sufficient to ensure good generalization. The model reached high training accuracy and low training loss, but validation accuracy plateaued early, and validation loss continued to rise, implying overfitting persisted.

These results may also be influenced by two key factors:

1. Restriction of 3 Hidden Layers: The limited depth of the neural network might have restricted the model's ability to learn complex patterns from the data, leading to suboptimal performance on the validation set. While 3 layers are often sufficient for simple tasks, more layers or an altered network architecture could potentially improve the model's capacity to generalize.

2. Dataset Limitation (20k Rows): The relatively small dataset (20,000 rows) could have contributed to the overfitting issue. With limited data, the model may have been able to memorize the training data without learning generalizable patterns. Expanding the dataset or employing techniques like data augmentation might help improve the model's generalization.

The model was evaluated on both the test set and the training set, producing notable insights into its performance and generalization capabilities. The results for both sets highlight significant gaps between the predicted answers and the true answers.

Test Set Results

The test set evaluation revealed substantial discrepancies between the **predicted** and **true** answers, which affected the model's BLEU and Levenshtein distance scores. Below are some specific observations:

- **BLEU Scores** were consistently low across all instances, with the **average BLEU score for the test set** being **0.0069**. This suggests that the model's predictions were far from the true answers, indicating that it struggled with understanding the exact phrasing of the questions.

- The **Levenshtein distance**, which measures the difference between the predicted and true answers, averaged **4.22** on the test set. This indicates that the predictions were, on average, several characters apart from the correct answers.

Examples of test set predictions:

- **Context**: North Carolina provides a large range of recreational activities.
  True Answer: recreational
  Predicted Answer: Sanskrit dramas
  BLEU Score: 0.0270
  Levenshtein Distance: 13

- **Context**: Luis Enrique returned to Barcelona as head coach.
  True Answer: Luis Enrique
  Predicted Answer: San Fathi
  BLEU Score: 0.0243
  Levenshtein Distance: 11

The low BLEU scores and high Levenshtein distances indicate a significant divergence between predicted and true answers.

Train Set Results

The evaluation on the **train set** produced slightly better results, with an **average BLEU score of 0.0105** and a **Levenshtein distance of 0.30**. This improvement is expected, as the model is more likely to perform better on data it has already encountered during training.

- **Train Set Example 1**:
  Context: The hindgut or proctodaeum in digestion.
  True Answer: the hindgut
  Predicted Answer: the hindgut
  BLEU Score: 1.0000
  Levenshtein Distance: 0

- **Train Set Example 2**:
  Context: Unicode developed with the International Organization for Standardization.
  True Answer: International Organization for Standardization
  Predicted Answer: International Energy Agency of Peoples
  BLEU Score: 0.3189
  Levenshtein Distance: 29

The model performed well on some questions, especially when the true answer was simple and directly matched the context. However, for more complex questions, the model struggled, as seen in the **Levenshtein distance** for some predictions.

Interpretation

- The **test set results** suggest that the model struggles with generalization, producing significantly different answers from the expected ones, particularly when the answer requires deeper contextual understanding.

- The **train set results** indicate that the model has memorized parts of the training data, as evidenced by perfect predictions in some cases, but still makes considerable errors for more complex questions.

The **high Levenshtein distance** in both sets suggests that the model may benefit from more refined architectures or training techniques that focus on improving the precision of answers.

2) Word level Bi-LSTM model.

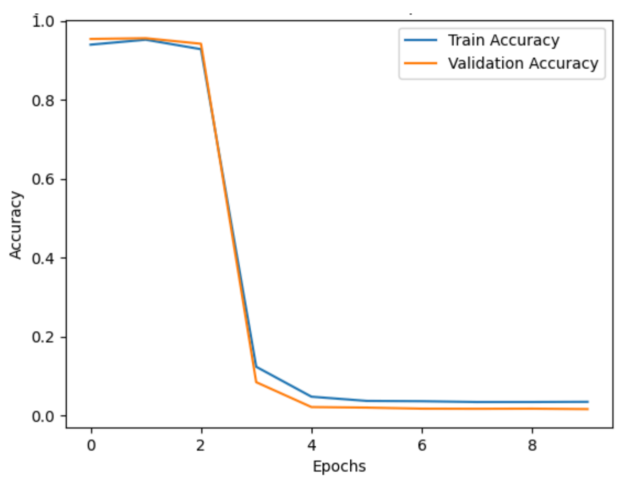
Preprocessing Techniques

We made the same preprocessing for this model too, but we made some differences, such as lower-casing the data and word level tokens using the tokenizer. We also added to new words in the vocab and in all the answers in the data, which are ‘<start>’ and ‘<end>’ to mark the beginning and end of sequences.

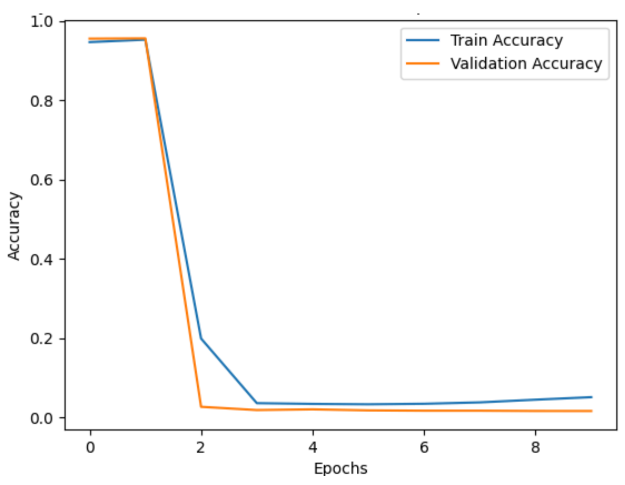
We only tried 2 different models, and we tried only two learning rates because we were limited to the GPU power and time for training as the word level takes longer training time and more memory to train, so we trained only 2 models with 2 different learning rates. Model parameters was:

1. **Optimizer:** We selected **Adam** optimizer.
2. **Batch Size**: We tested batch sizes of **32**. We couldn’t reduce or increase the batch size because less than that train might be unstable and more than that the memory will be full, so we couldn't train on more than 32.
3. **Number of Dense Units**: We used dense layer size equal to the **vocabulary size**.
4. **Learning Rate**: Two learning rates were evaluated: **0.001** and **0.0005**. A learning rate of **0.001** provided better late accuracies, and
5. These are the accuracy graphs from the models trained:

Model with LR = 0.0005



Model with LR = 0.001

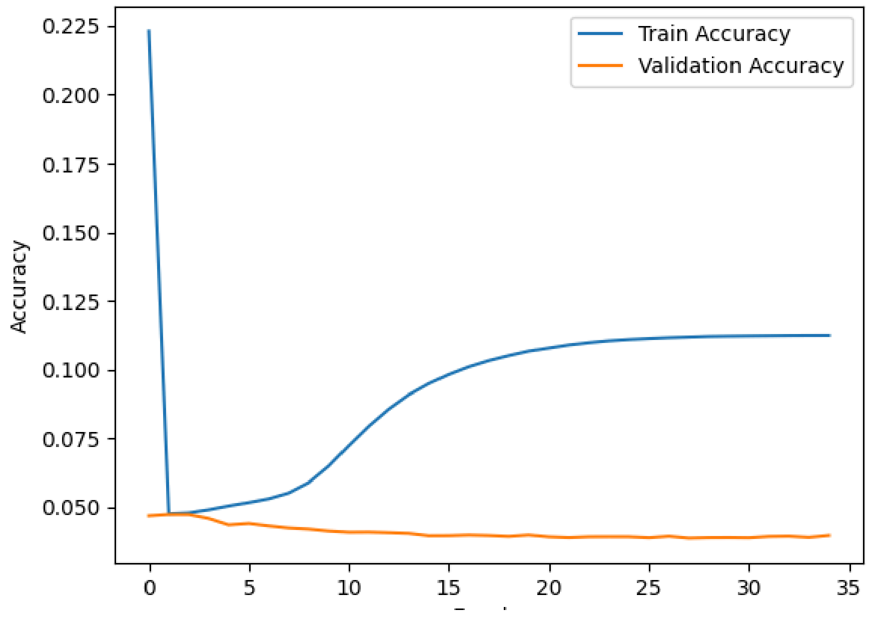


We found that the accuracy started to be 95%+ at the start, then it dropped significantly, we found a possible explanation

1. Initial High Accuracy (95%): This is likely due to overfitting. This creates an artificially high accuracy that doesn't reflect true performance.

2. Drop to 3%: Once the model starts learning the actual patterns in the data, performance drops dramatically. This represents the model's true baseline performance on the actual task.

Then we tried adding <start> and <end> to the answer sequence at our final model, and we observed that the accuracy started to be more realistic, at the start, as you can see at this graph:



This possibility happens because the `<start>` token provides a clear beginning point. The model has clearer sequence boundaries. The task is slightly more structured.

Findings

The model was trained over 35 epochs using an Bi-LSTM architecture with cuDNN acceleration enable. Throughout the training process, several important trends were observed in the accuracy and loss metrics for both training and validation sets.

Accuracy and Loss Trends

- Training Accuracy steadily increased from 4.7% in epoch 2 to a maximum of 11.25% by epoch 34, indicating that the model was able to learn patterns from the training data.
- Validation Accuracy, however, it was decreasing from maximum 4.6% till it became 3.9% at the end, this is possibly because the model could not generalize as the vocab size were huge, and The model has to learn patterns from a very large vocabulary.
- Training Loss consistently improved, dropping from 6.47 in epoch 1 to 0.0153 by epoch 35, showing that the model minimized error on the training set.
- In contrast, Validation Loss increased steadily from 5.74 in epoch 1 to over 10.38 by epoch 35, suggesting limited generalization.

Score Summary (Selected Epochs)

Epoch	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
2	4.77%	4.74%	5.12	5.79
14	9.09%	4.05%	1.21	8.22
35	11.25%	3.97%	0.01	10.38

Interpretation

These results may be influenced by the same reasons we mentioned in the character LSTM model, which is the 3 hidden layer restriction and the 20k rows limitation.

The model was evaluated on both the test set and the training set, producing notable insights into its performance and generalization capabilities. The results for test set highlight significant gaps between the predicted answers and the true answers. But for the training set, the model answered every question right.

Test Set Results

The test set evaluation revealed substantial discrepancies between the **predicted** and **true** answers, which affected the model's BLEU and Levenshtein distance scores. Below are some specific observations:

- **BLEU Scores** were consistently low across all instances, with the **average BLEU score for the test set** being **0.0071**. This suggests that the model'spoor generation.

- The **Levenshtein distance**, which measures the difference between the predicted and true answers, averaged **3.59** on the test set. This indicates that the answers generated were, on average, several words apart from the true answers.

Examples of test set predictions:

- **Context :** The Atlantic Ocean has less influence on the climate of the Piedmont region, which has hotter summers and colder winters than in the coast**.**
  Question : What region of North Carolina has hotter summers and colder winters than the coast?
  True Answer: Piedmont
  Predicted : european orthodox
  BLEU Score : 0.0223
  Levenshtein Distance: 14

- **Context :** Early humanists saw no conflict between reason and their Christian faith (see Christian Humanism).
  Question : Who was able to reconcile their religious beliefs with those of humanism?
  True Answer: Early humanists
  Predicted : technical over one power
  BLEU Score : 0.0355
  Levenshtein Distance: 22

The low BLEU scores and high Levenshtein distances indicate a significant divergence between predicted and true answers.

Train Set Results

The evaluation on the **train set** produced better results, with an **average BLEU score of 0.0166** and a **Levenshtein distance of 0.02**. This improvement is expected, as the model is more likely to perform better on data it has already encountered during training.

- **Train Set Example 1**:
  **Context : ** The evidence for the effectiveness of measures to prevent the development of asthma is weak.
  **Question : ** What is weak that is not helping prevent the development of asthma?
  True Answer: The evidence for the effectiveness of measures to prevent
  **Predicted : ** the evidence for the effectiveness of measures to prevent
  **BLEU Score : ** 0.9820
  Levenshtein Distance: 1

- **Train Set Example 2**:
  Context : Around 746, Abu Muslim assumed leadership of the Hashimiyya in Khurasan.
  Question : Who became leader of the Khurasan Hashimiyya in approximately 746?
  True Answer: Abu Muslim
  Predicted : abu muslim
  BLEU Score : 0.5254
  Levenshtein Distance: 2

The model performed well on all questions. However, the results are bad, although the answer is correct this is because the lowercase preprocessing we made, which is unnecessary, the model should produce higher case and lower case normally as it may differ the meaning. So we need to train again with normal cases.

Limitations

While the model was evaluated on both the training and test sets, several limitations were identified that may have contributed to the suboptimal performance, particularly on the test set. These limitations can be broadly categorized into the following areas:

1. Limited Dataset Size

One of the primary limitations of the model is the **size of the training dataset**. With only 20,000 rows, the dataset is relatively small for training a deep learning model, particularly one that aims to handle diverse and complex question-answer pairs. A larger and more diverse dataset would likely help the model generalize better, reducing the **Levenshtein distance** and improving the **BLEU score**. The small dataset may have caused the model to overfit to certain patterns, which, while benefiting performance on the training set, led to poor generalization on the test set.

2. Limited Model Architecture (Number of Layers)

The model architecture used for training includes only **3 hidden layers**, which may not be deep enough to capture complex patterns in the data. Models with deeper architectures (i.e., more layers) have the potential to learn more intricate relationships between input and output, but this requires more computational resources and time for training. With limited layers, the model might have struggled to accurately process and generate answers for complex or nuanced questions, leading to a high **Levenshtein distance** and low **BLEU scores**.

3. Time Constraints

Given the resources available for training, the model's training time may have been insufficient for it to converge optimally. Deep learning models, especially those working with text data, often require significant amounts of training time to fine-tune the weights and achieve better accuracy. The limited training time, coupled with the relatively small dataset, may have hindered the model from fully learning the underlying patterns and producing more accurate predictions.

4. Resource Limitations

The computational resources available during training (e.g., processing power, memory) might have been limited, restricting the model's ability to process the data effectively. Deep learning models, particularly those with more layers and complex structures, require substantial computational resources to train efficiently. The lack of such resources could have impacted the model's performance, especially in terms of processing larger datasets or training more complex architectures.

Potential Improvements

To address the above limitations and improve the model's performance, several strategies could be implemented:

1. Expanding the Dataset

Increasing the size and diversity of the training dataset is one of the most straightforward ways to improve the model's generalization capabilities. A larger dataset, ideally with more examples of varied question-answer pairs, would allow the model to learn better representations of the relationships between different types of questions and answers. Additionally, incorporating domain-specific data could help the model understand the context more effectively.

2. Increasing Model Depth (More Layers)

To improve the model's ability to capture complex patterns, increasing the **number of layers** in the neural network would be beneficial. Deeper architectures can learn more abstract and complex features, which is essential for tasks such as question answering. Using **convolutional neural networks (CNNs)**, **recurrent neural networks (RNNs)**, or more complex architectures like **transformers** can help achieve this. However, increasing the depth would also require more computational resources and time.

3. Attention Mechanisms and Transformers

A major improvement could come from using **attention mechanisms** or **transformer-based models**. Attention mechanisms, such as those used in **BERT**, **GPT**, or **T5**, allow the model to focus on different parts of the input sequence more effectively, improving its ability to generate accurate answers by attending to relevant context. **Transformers** are especially useful for natural language processing tasks because they enable parallel processing of input sequences, making them more efficient and effective at capturing long-range dependencies in text.

- **BERT** (Bidirectional Encoder Representations from Transformers) and similar models could allow the model to better understand the context of questions by considering words in both directions (left and right), instead of just processing them sequentially.

- **GPT** (Generative Pretrained Transformer) models, which are autoregressive and focus on generating text, could also help improve performance in generating answers.

Implementing such architectures would require significant changes to the model's design, but they could greatly enhance its performance.

4. Data Augmentation and Preprocessing

Another potential improvement involves **data augmentation** and more advanced **preprocessing techniques**. Data augmentation methods, such as paraphrasing questions or adding noise to the data, could help increase the robustness of the model. Additionally, more sophisticated **tokenization** and **lemmatization** techniques could improve how the model handles words and phrases, further boosting its accuracy.

5. Regularization and Fine-Tuning

Incorporating better **regularization** methods, such as **dropout**, **L2 regularization**, or **early stopping**, could help prevent overfitting to the training data, which would improve generalization to the test set. Additionally, fine-tuning pre-trained models (e.g., fine-tuning BERT on the specific dataset) could significantly improve performance, as the model would start with weights learned from large corpora and then adapt to the specific task.

Conclusion

While the current model faced several limitations—such as limited dataset size, architectural constraints, and resource limitations—there are multiple strategies that could be employed to improve its performance. Expanding the dataset, increasing model depth, implementing attention mechanisms, and leveraging advanced preprocessing techniques are all potential avenues for improvement. By addressing these limitations, it is likely that the model would perform better on both the training and test sets, achieving more accurate predictions with lower **Levenshtein distances** and higher **BLEU scores**.