A Deep Learning Approach to QA Using the SQuAD Dataset

Aly Raafat, Karim Mohamed, Omar Ahmed

The German University in Cairo

Abstract. This project presents a deep learning-based question answering (QA) system that finds answers from a given piece of text. The system takes a natural language question and a related context, and it learns to locate the correct answer within that context. We use a simplified version of the SQuAD 2.0 dataset, which includes 20,000 examples where answers are available. The model is limited to three internal layers (not counting the input and output layers), and we test different types of neural networks, such as RNNs, LSTMs, GRUs, and attention-based models. This paper describes how we prepared the dataset, built and trained the models, and evaluated their performance. We also discuss what worked well, the challenges we faced, and how the system could be improved in the future.

Keywords: Question Answering, SQuAD 2.0, Deep Learning, LSTM, Attention.

1 Introduction

Question Answering (QA) is an important task in Natural Language Processing (NLP), where a system tries to find the answer to a question based on a given text (called the context). QA models are used in many real-world applications like search engines, virtual assistants, and chatbots. In this project, we focus on extractive question answering, where the answer must be a continuous span of text taken directly from the context.

We use a subset of the SQuAD 2.0 dataset, which is one of the most widely used benchmarks for QA tasks. It was one of several datasets suggested for this project, and we chose it because of its popularity and relevance to real-world QA problems. We filtered the dataset to include only 20,000 answerable question-context-answer examples.

As a starting point, we experimented with a simple setup where the model tries to answer questions without using any context. As expected, the model struggled to perform well, which confirmed the importance of including context information in extractive QA systems.

We implemented multiple models using PyTorch, including LSTM-based, GRU-based, and attention-based architectures. We used the HuggingFace ‘tokenizers‘ library for efficient text preprocessing. Our model designs were restricted to a maximum of three layers (excluding input and output), and we were instructed to use either RNNs or attention-based models,but not a combination of both. However, through a brief literature review, we found that combining RNNs with attention mechanisms often gives better results, especially in small-scale models like ours.

Another challenge was the limited computational resources available, which made it difficult to experiment with larger models or perform many training iterations. Despite these constraints, we aimed to build a functional and reasonably accurate QA system.

This report describes how we processed the data, built and trained the models, evaluated their performance, and what we learned from the process. We also discuss the limitations of our approach and propose directions for future improvements.

2 Literature Review

The Stanford Question Answering Dataset (SQuAD), introduced by Rajpurkar et al. (2016), has been a central benchmark for evaluating extractive question answering (QA) systems. It consists of over 100,000 questions created from Wikipedia articles, where the task is to extract the correct span from a given context. The dataset’s design has encouraged a range of model architectures over the years,starting with recurrent neural networks (RNNs) and gradually moving toward attention-based models and transformer architectures.

2.1 RNN-based Approaches

Initial attempts to solve the SQuAD task mainly used RNNs like LSTMs and GRUs due to their ability to model sequential dependencies. For example, the Match-LSTM model by Wang and Jiang (2016) used an LSTM to align the question and context, followed by a pointer network to predict the start and end of the answer span. The idea was to compute attention-weighted representations of the passage conditioned on the question, which helped the model focus on relevant information.

Another early model, BiDAF [1], also relied on LSTMs but introduced a bi-directional attention mechanism that allowed interaction in both directions,context to question and question to context. This helped the model generate more informed representations of the input, leading to better span predictions.

Bidirectional GRUs have also been used as a lighter alternative to LSTMs. Mishra (2022), for example, proposed a model based on Bi-GRUs with attention layers for SQuAD 2.0, which includes unanswerable questions. Their architecture could both predict answer spans and assess whether a question has an answer at all, although exact performance scores weren't reported.

In another example, Hu et al. (2018) proposed a model that combined Bi-LSTMs with co-attention mechanisms and a boundary decoder for span prediction. Their model performed competitively and showed how combining attention with recurrent encoders could boost accuracy.

2.2 Attention-Based and Transformer Models

As attention mechanisms became more central in NLP, models began to rely more on them, reducing or even eliminating the need for recurrence. While models like BiDAF already used attention, the introduction of transformers made it possible to model dependencies across the entire sequence in parallel, using only self-attention.

One of the most impactful transformer-based models is BERT (Devlin et al., 2018). BERT is a pre-trained transformer that was fine-tuned on SQuAD, and it quickly set new benchmarks. Its ability to model bidirectional context using self-attention proved very effective for extractive QA tasks. BERT achieved significantly higher scores than previous RNN-based models and has since become a strong baseline in the field.

Since BERT, many improved models have been proposed. These include larger or more specialized transformer architectures, often using techniques like ensemble methods, knowledge distillation, or adversarial training. Many of these models have surpassed human-level performance on the SQuAD leaderboard, especially in SQuAD 1.1 and 2.0.

2.3 Summary

The progression of QA models on SQuAD reflects a clear shift from sequence-based models like LSTMs and GRUs to attention-dominant architectures and finally to transformers. While our project was restricted to using either RNNs or attention-based models,not both,the literature suggests that combining these techniques can lead to improved performance, particularly for smaller models. This context helped inform the design choices in our own model development.

3 Data Understanding & Preprocessing

Effective data preprocessing was a crucial part of our pipeline to ensure the model could handle the input format correctly and learn from the SQuAD 2.0 dataset under our project constraints.

3.1 Data Understanding

Figure 1 shows that most answers in the dataset are very short. The majority of answers are between 1 to 3 words long, with the highest frequency at length 1. The frequency drops quickly as the answer length increases, and answers longer than 10 words are relatively rare. This indicates that short answers dominate the dataset. This is why we filtered only small answers as they are the majority of the dataset.

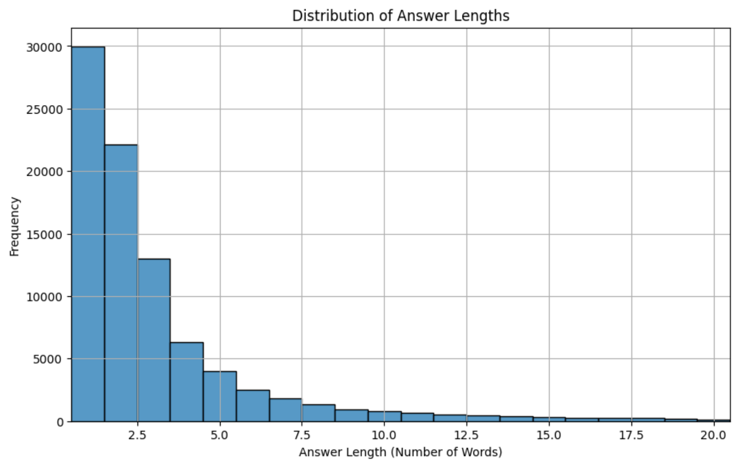

Fig. 1. Histogram for the answer distribution

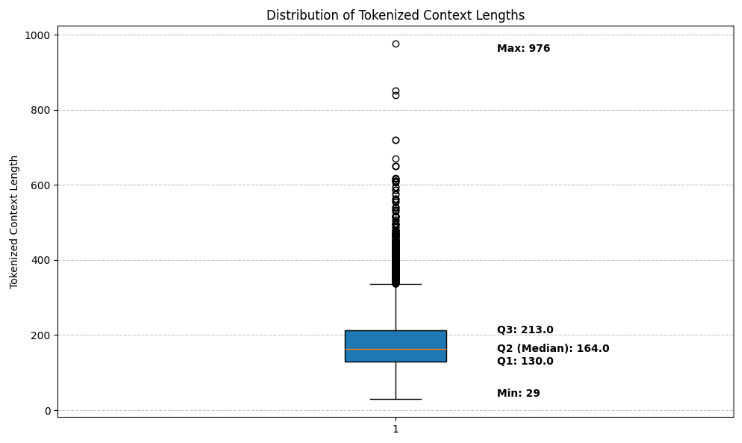

Fig. 2. boxplot for the tokenized context length distribution

Figure 2 shows that most tokenized contexts are between 130 and 213 tokens, with a median length of 164. The shortest context is 29 tokens long, while the longest reaches 976 tokens. A significant number of outliers exceed the upper quartile, with 707 samples marked as outliers. The 95th, 97th, and 99th percentiles are 318, 345, and 417 tokens respectively, indicating that most contexts remain well below the maximum input limit of common transformer models.

We also examined the lengths of question, answer and the combined lengths of context + question tokens. To ensure efficient processing while minimizing data loss, we chose the 95th percentile as a cutoff point for truncation across all input types. This percentile-based strategy allows us to retain the majority of samples without exceeding model constraints.

After analyzing the dataset and filtering for answerable questions, we found that it contains 18,877 unique contexts, each linked to one or more question–answer pairs. This is significant as it can help us reduce overfitting.

3.2 Dataset Preparation

We used the SQuAD 2.0 dataset, which includes both answerable and unanswerable questions. For our task, we focused only on answerable questions. The dataset was downloaded in JSON format from the official website and parsed using custom Python scripts.

To reduce training time and memory usage, we sorted all answerable examples by answer length and selected the 20,000 shortest ones. Each example included a question, a context, and the ground truth answer span (given as character-level start and end positions within the context).

3.3 Tokenization

We used **Byte Pair Encoding (BPE)** to tokenize the text. A vocabulary of 10,000 subword tokens was generated from the dataset using HuggingFace's `tokenizers` library. This was appropriate given that the dataset had approximately 90,000 unique raw words. BPE allowed us to break rare or unknown words into more frequent subword units, reducing the effective vocabulary size while still capturing meaningful tokens. We also tested with **Word Level Tokenization** so we can use **GloVe** pre-trained embeddings.

All text was lowercased as part of preprocessing. We avoided removing punctuation or other characters, since the ground truth answer spans in SQuAD are given as character-level indices. Any modification of the text could invalidate these indices and cause incorrect training labels.

3.4 Input Construction

Each input sequence was constructed by concatenating the tokenized question and context with special tokens in the following format:
`[SOS] question tokens [SEP] context tokens [SEP]`

We used the following special tokens throughout preprocessing:

- `[UNK]` — Represents unknown tokens not in the vocabulary.
- `[PAD]` — Used to pad shorter sequences to match a fixed length. These tokens are masked out during training.
- `[MASK]` — Included in the vocabulary for completeness, though not used in this task.
- `[SOS]` — Marks the beginning of a sequence.
- `[EOS]` — Marks the end of a sequence. Not strictly needed for extractive QA, but included for consistency.
- `[SEP]` — Separates the question and the context in the input sequence.

For example, a sample input could look like:
`[SOS] What is the capital of France? [SEP] France is a country in Europe. Its capital is Paris. [SEP]`

3.5 Truncation and Padding

Inputs longer than the model's maximum allowed sequence length were **truncated** from the end of the context. This was necessary due to GPU memory constraints and to maintain a consistent input size. Shorter sequences were **padded** with the `[PAD]` token.

To avoid the model attending to these padded positions during training and evaluation, we applied **attention masks**. These masks assign a weight of 0 to `[PAD]` tokens and 1 to all valid tokens, ensuring the model only learns from meaningful parts of the input.

3.6 Label Processing

Since the original dataset provides **character-level start and end indices** of the answer in the context, we had to convert these into **token-level indices** after tokenization.

This was done by:

1. Tokenizing the context separately.
2. Mapping the character-based start and end positions of the answer to the corresponding tokens.
3. Adjusting the indices to align with the combined `[SOS] question [SEP] context` input.

During training, the model learns to predict these token indices as the start and end positions of the answer.

3.7 Dataset Splits and Batching

We used the original **training and development (dev)** splits provided in SQuAD 2.0 and applied our filtering strategy to both sets to keep only answerable examples. These filtered sets were then used for training and validation respectively.

We trained using a **batch size of 32 or 64**, and all sequences in a batch were padded to the same length to enable efficient parallel computation. Padding masks were used to ensure correct attention and loss computation.

To efficiently handle the data during training, we implemented a **custom PyTorch Dataset class** that inherits from `torch.utils.data.Dataset`. This class handles preprocessing on-the-fly and returns the model inputs and labels in a consistent format. We then used **PyTorch DataLoaders** to create batches, shuffle the training data, and manage parallel data loading during training and evaluation.

3.8 Libraries and Tools

We used the following tools in our preprocessing pipeline:

– **HuggingFace Tokenizers** — For BPE-based tokenization and handling special tokens.
– **Custom Python scripts** — For loading and filtering the dataset, mapping character indices to token indices, and constructing input sequences with proper masks.
– **PyTorch Dataset and DataLoader** — For batching, shuffling, and parallelized data loading.

4 Model Architectures

In this section, we present the five neural network architectures we implemented for extractive question answering on the SQuAD dataset. Each model receives a tokenized pair of question and context sequences and is trained to predict the start and end indices of the answer span within the context. All models share a common data preprocessing pipeline and evaluation setup, ensuring fair comparison across architectures. The implemented models span a range of design paradigms, including RNN-based encoders, attention mechanisms, and fully transformer-based models. These architectural variations allow us to analyze the trade-offs between efficiency, complexity, and accuracy in span-based question answering tasks. We categorize our models into two main families:

– **RNN-based models**: These include the *Dual-RNN*, *DrQA-inspired BiLSTM*, and *BiDAF* architectures, each differing in how they encode inputs and model interactions between the question and context.

– **Transformer-based models**: These include the *Transformer-Encoder QA Model with GloVe Embeddings*, which leverage self-attention to capture token-level dependencies and richer contextual relationships.

Below, we describe each model in detail, highlighting the architectural components, input/output structure, and distinguishing features.

4.1 Dual-RNN with Final State Concatenation

This model processes the question and context independently using two separate RNN encoders. Unlike attention-based models, it summarizes each input sequence using the final hidden state of its respective RNN. The final hidden states of both the question and context are concatenated and passed through a feed-forward network to predict the start and end positions.

Architecture Overview

– **Separate Embedding Layers**: Independent embeddings for the context and question inputs. Optionally initialized with pretrained vectors and can be frozen during training.
– **Dual RNN Encoders**: Separate RNN (LSTM/GRU/RNN) modules for the context and question. The outputs of these encoders are not used directly,only their final hidden states are kept.
– **Hidden State Concatenation**: The final forward and backward hidden states (if bidirectional) from both RNNs are concatenated.
– **Feedforward Layers**: A two-layer MLP uses the combined representation to produce logits for the start and end positions.

Input & Output

– **Inputs**:
  - `context (LongTensor)`: Tokenized context sequence.
  - `question (LongTensor)`: Tokenized question sequence.
– **Outputs**:
  - `start_logits`, `end_logits`: Logits over the context, computed from the combined hidden states.

4.2 Cross-Attention Transformer Model

This model adopts an approach inspired by encoder-decoder architectures, specifically focusing on enriching the context representation by attending to the question.

Architecture Overview

- **Embedding Layer**: Context and question tokens are independently embedded with token and positional embeddings. Dropout is applied.
- **Question Encoder**: Embedded question is passed through a Transformer Encoder block.
- **Cross-Attention Layers**: Context representations act as queries, and the encoded question representations act as keys and values.
- **Output Heads**: The final context representations are passed through linear layers to produce start and end logits.
- **Masking**: Padding tokens are masked in both context and question during attention computations.

Input & Output

- **Inputs**:
  - `context (LongTensor)`
  - `question (LongTensor)`
  - `attention_mask_context (Tensor)` (optional)
  - `attention_mask_question (Tensor)` (optional)
- **Outputs**:
  - `start_logits`, `end_logits`

4.3 DrQA-Inspired BiLSTM Model [2]

This model is a simplified version of the DrQA Document Reader. It uses separate BiLSTM encoders for the question and context, followed by a dot-product attention mechanism and a pointer network.

Architecture Overview

- **Embedding Layer**: Maps tokens to dense vectors; handles padding.
- **BiLSTM Encoders**: Encode context and question independently with 3-layer BiLSTM networks.
- **Dot-Product Attention**: Computes a similarity matrix between context and question encodings.
- **Fusion Layer**: Concatenates attention-weighted question vectors with context representations.
- **Pointer Network**: Uses MLPs to predict start and end positions.

Input & Output

- **Inputs**:
  - `context (LongTensor)`
  - `question (LongTensor)`
  - `attention_mask_question (Tensor)` (optional)
- **Outputs**:
  - `start_logits`, `end_logits`

4.4 Bi-Directional Attention Flow (BiDAF) Model [1]

The BiDAF model explicitly models the interaction between context and question using bi-directional attention mechanisms.

Architecture Overview

- **Embedding Layer**: Converts word indices to vectors using an embedding layer (optionally pretrained).
- **Contextual Encoder**: BiLSTM encodes context and question sequences independently.
- **Attention Flow Layer**:
  - **Context-to-Query (C2Q)** Attention
  - **Query-to-Context (Q2C)** Attention
  - Combines original and attended vectors to form augmented representation G.
- **Modeling Layer**: 2-layer BiLSTM processes G to produce refined representation M.
- **Output Layer**: Combines G, M, and M_2 to compute logits via linear layers.

Input & Output

- **Inputs**:
  - `context (Tensor)`: (batch, c_len)
  - `question (Tensor)`: (batch, q_len)
- **Outputs**:
  - `start_logits`, `end_logits`

4.5 Transformer-Encoder QA Model with GloVe Embeddings

We implemented a transformer-based question answering model, `TransformerQAModel3`, tailored for extractive QA tasks. This architecture encodes a concatenated sequence of the question and context and predicts the start and end positions of the answer span within the context.

Architecture Overview

- **Embedding Layer**: The model begins with a token embedding layer initialized using pretrained GloVe embeddings. These embeddings provide semantically rich representations and can optionally be frozen during training.
- **Positional Encoding**: Learnable positional embeddings are added to the token embeddings to provide the model with positional information, which transformers lack inherently.

- **Transformer Encoder**: The main component of the model is a stack of Transformer encoder layers. Each layer includes multi-head self-attention and a feed-forward network, along with layer normalization and residual connections, allowing the model to capture global contextual relationships across the entire sequence.
- **Dropout Layer**: Dropout is applied to both the embeddings and transformer outputs to reduce overfitting during training.
- **Prediction Heads**: Two independent linear layers compute logits for the start and end positions of the answer. These operate over the final encoded representation of the context tokens only.

Input & Output

- **Inputs**:
  - `context_question (LongTensor)`
  - `attention_mask_context_question (Tensor)` (optional)
- **Outputs**:
  - `start_logits`, `end_logits`

5 Training Procedure

5.1 Training Configuration

All four models were trained using the same configuration for consistency and fair comparison. The training setup is as follows:

- **Optimizer:** AdamW
- **Learning Rate:** 0.001
- **Batch Size:** 32 or 64, depending on memory availability
- **Number of Epochs:** 20
- **Dropout Rate:** 0.1
- **Early Stopping:** Training was stopped early if no improvement in validation loss was observed for 5 consecutive epochs.
- **Learning Rate Scheduler:** The learning rate was reduced by a factor of 0.1 if validation performance did not improve for 2 consecutive epochs.

5.2 Loss Function

We used the Cross-Entropy Loss to train all models, implemented as `nn.CrossEntropyLoss` in PyTorch. Since the task is formulated as predicting the start and end token positions of the answer span within the context, we compute the cross-entropy loss separately for both the start and end positions. The final loss is the sum of the two individual losses:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{start}} + \mathcal{L}_{\text{end}} \tag{1}$$

This approach treats the prediction of the start and end indices as two separate multi-class classification tasks over the sequence length of the context.

5.3 Evaluation Metrics

To evaluate our models, we used a variety of metrics that assess performance at both the token and span levels. Each of the following metrics is computed independently for the **start** and **end** positions of the predicted answer span, allowing us to monitor their convergence during training:

– **Start Accuracy & End Accuracy:** These metrics measure the proportion of correct predictions for the start and end positions, respectively. Monitoring both allows us to observe how well the model learns to identify each boundary of the answer span.
– **Start Precision & End Precision:** Precision is computed separately for start and end positions as the fraction of predicted positions that match the true start or end token.
– **Start Recall & End Recall:** Recall reflects how many of the true start or end tokens were correctly predicted. This metric is also tracked individually for both positions.
– **Start F1 Score & End F1 Score:** The harmonic mean of precision and recall, providing a balanced measure of model performance for both the start and end token predictions.
– **Joint Exact Match (EM):** This metric indicates the percentage of predictions where *both* the start and end positions exactly match the ground truth span. It is a strict metric for measuring exact span extraction accuracy.
– **Span Overlap F1:** A more lenient metric that measures the F1 score based on the overlap between the predicted and ground truth spans, treating both as sets of tokens. This metric captures partial correctness in span prediction.

All metrics were tracked across training epochs to assess convergence and guide model selection.

6 Evaluation

We evaluate our models using standard metrics for extractive question answering: start and end accuracy, precision, recall, F1-score, joint exact match (EM), and span overlap F1. The results for both training and testing datasets are presented in the following tables.

6.1 Results

As shown, the BiDAF and DrQA models significantly outperform the others in both training and validation metrics. BiDAF achieves high training performance, maintaining decent generalization on the validation set with a joint exact match (EM) of 0.45 and span F1 of 0.52. DrQA performs slightly better on training but generalizes slightly worse than BiDAF.

In contrast, the Glove Transformer and RNN-based models (especially the Context Separated variant) perform poorly, indicating issues in either architectural choices or their ability to capture relationships in the context-question span prediction task. The QA Separated Transformer shows some promise but still lags behind the more traditional attention-based architectures like BiDAF.

This reinforces the effectiveness of combining attention with recurrent mechanisms (as in BiDAF), particularly in tasks requiring precise span detection like extractive QA with small scale models like our case.

Table 1. Training evaluation metrics for all models

Model	**Start Acc**	**End Acc**	**Start F1**	**End F1**	**Start Prec**	**End Prec**	**Start Rec**	**End Rec**	**Joint EM**	**Span F1**
BiDAF	0.83	0.83	0.83	0.83	0.84	0.84	0.83	0.83	0.80	0.83
DrQA	0.84	0.83	0.84	0.83	0.84	0.84	0.84	0.83	0.80	0.82
GloVe Transformer	0.26	0.26	0.26	0.27	0.26	0.27	0.26	0.26	0.22	0.23
QA Transformer Sep.	0.18	0.20	0.17	0.19	0.19	0.22	0.18	0.20	0.09	0.12
RNN Context Sep.	0.06	0.05	0.02	0.02	0.02	0.02	0.06	0.05	0.03	0.06

Table 2. Testing evaluation metrics for all models

Model	**Start Acc**	**End Acc**	**Start F1**	**End F1**	**Start Prec**	**End Prec**	**Start Rec**	**End Rec**	**Joint EM**	**Span F1**
BiDAF	0.52	0.51	0.52	0.51	0.54	0.54	0.52	0.51	0.45	0.52
DrQA	0.28	0.28	0.28	0.27	0.30	0.29	0.28	0.28	0.24	0.27
GloVe Transformer	0.13	0.13	0.13	0.13	0.14	0.15	0.13	0.13	0.11	0.12
QA Transformer Sep.	0.11	0.11	0.10	0.10	0.12	0.12	0.11	0.11	0.05	0.08
RNN Context Sep.	0.03	0.03	0.01	0.01	0.01	0.01	0.03	0.03	0.01	0.04

6.2 Qualitative Examples of Correct Predictions

Below are some selected examples from the validation set.

RNN Seperated

- **Question:** *How many Khitan tumens were there?*
- **Context (excerpt):** *... there were 4 Han tumens and 3 Khitan tumens, with each tumen consisting of 10,000 troops ...*
- **Predicted Answer:** `3`
- **True Answer:** `3`

– **Question:** *What is the atomic number of the element oxygen?*
– **Context (excerpt):** *... oxygen is a chemical element with symbol O and atomic number 8 ...*
– **Predicted Answer:** `and atomic`
– **True Answer:** `8`

QA Transformer Seperated

– **Question:** *How many Khitan tumens were there?*
– **Context (excerpt):** *... there were 4 Han tumens and 3 Khitan tumens, with each tumen consisting of 10,000 troops ...*
– **Predicted Answer:** `3`
– **True Answer:** `3`

– **Question:** *What is the atomic number of the element oxygen?*
– **Context (excerpt):** *... oxygen is a chemical element with symbol O and atomic number 8 ...*
– **Predicted Answer:** `20.8%`
– **True Answer:** `8`

GloVe Transformer

– **Question:** *How many Khitan tumens were there?*
– **Context (excerpt):** *... there were 4 Han tumens and 3 Khitan tumens, with each tumen consisting of 10,000 troops ...*
– **Predicted Answer:** `3`
– **True Answer:** `3`

– **Question:** *What is the atomic number of the element oxygen?*
– **Context (excerpt):** *... oxygen is a chemical element with symbol O and atomic number 8 ...*
– **Predicted Answer:** `oxygen`
– **True Answer:** `8`

BiDAF

– **Question:** *How many Khitan tumens were there?*
– **Context (excerpt):** *... there were 4 Han tumens and 3 Khitan tumens, with each tumen consisting of 10,000 troops ...*
– **Predicted Answer:** `3`
– **True Answer:** `3`

– **Question:** *What is the atomic number of the element oxygen?*
– **Context (excerpt):** *... oxygen is a chemical element with symbol O and atomic number 8 ...*
– **Predicted Answer:** `8`
– **True Answer:** `8`

DrQA

- **Question:** *How many Khitan tumens were there?*
- **Context (excerpt):** *... there were 4 Han tumens and 3 Khitan tumens, with each tumen consisting of 10,000 troops ...*
- **Predicted Answer:** `two`
- **True Answer:** `3`

- **Question:** *What is the atomic number of the element oxygen?*
- **Context (excerpt):** *... oxygen is a chemical element with symbol O and atomic number 8 ...*
- **Predicted Answer:** `8`
- **True Answer:** `8`

All models predicted the correct answer for the *Khitan tumens* question except **DrQA**, which predicted `two` instead of `3`. All models failed on the *atomic number of oxygen* question except **BiDAF** and **DrQA**, which predicted the correct answer `8`.

6.3 Learning Curves

RNN Separated Figure 3 shows the training and validation loss over 9 epochs. The training loss keeps going down, which means the model is learning from the training data.

However, the validation loss starts increasing after around the 4th epoch. This means the model is starting to overfit , it performs well on training data but worse on unseen data.

This training was originally planned for 20 epochs, but it stopped early because of early stopping. Dropout and learning rate scheduling were also used to reduce overfitting, but the model still started to overfit after a few epochs. This suggests the model may need further tuning or regularization.

QA Transformer Separated Figure 4 shows the training and validation loss over 6 epochs. The training loss decreases steadily, which means the model is learning well on the training data.

However, the validation loss increases after the second epoch and becomes much higher than the training loss. This suggests the model starts to overfit very early , it memorizes the training data but does not perform well on new data.

Even though early stopping and other techniques were likely used, the gap between training and validation loss shows the model struggles to generalize. This might be due to a small dataset, high model complexity, or not enough regularization.

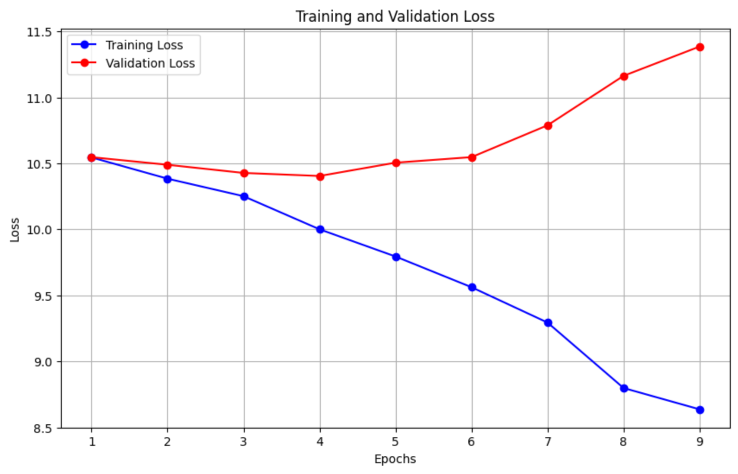

Fig. 3. Learning curves for the RNN Separated model showing training and validation loss.

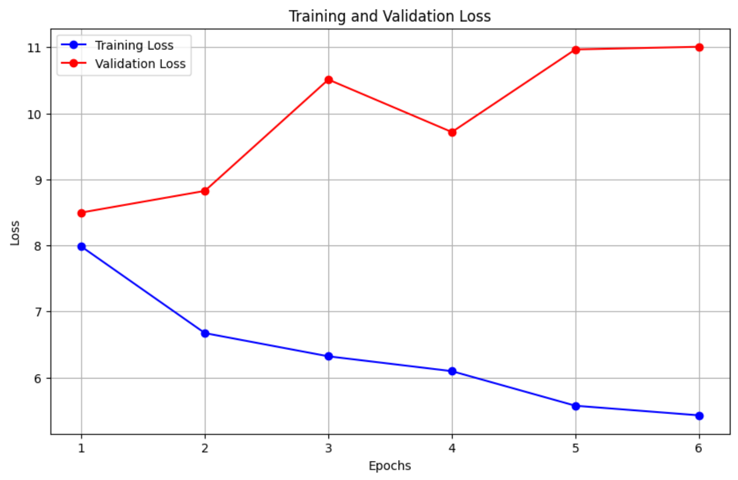

Fig. 4. Learning curves for the QA Transformer Separated model showing training and validation loss.

GloVe Transformer Figure 5 illustrates the training and validation loss over 10 epochs. The training loss shows a consistent downward trend, indicating that the model is learning effectively on the training data.

In contrast, the validation loss fluctuates throughout training and does not show clear improvement. After a slight decrease in the early epochs, it remains relatively high and even increases in later epochs, suggesting that the model is overfitting , it continues to fit the training data better while failing to improve on unseen data.

This pattern indicates poor generalization and may be due to insufficient regularization, overly complex architecture, or limitations in the dataset size or diversity. Despite the extended training duration, the persistent gap between training and validation losses emphasizes the need for strategies like stronger regularization or simpler models.

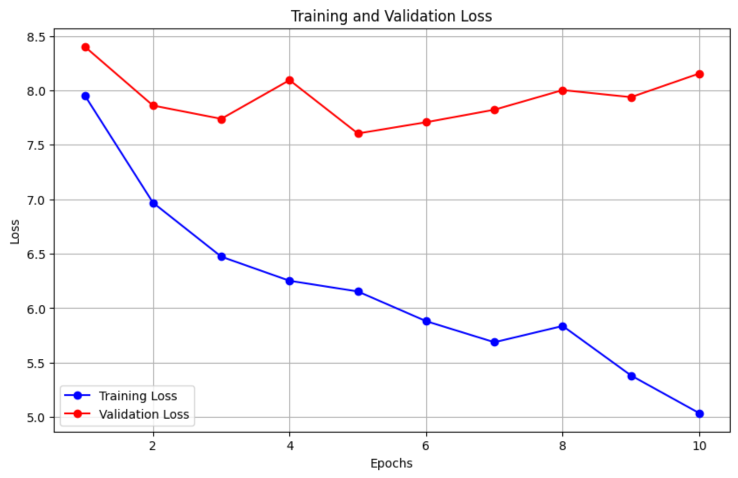

Fig. 5. Learning curves for the GloVe Transformer model showing training and validation loss.

BiDAF Figure 6 presents the training and validation loss over 8 epochs. The training loss shows a sharp and consistent decline, suggesting that the model is effectively minimizing error on the training set.

In contrast, the validation loss initially decreases slightly but then begins to rise steadily from the fourth epoch onward. By the final epochs, the gap between training and validation loss becomes significant, indicating that the model is overfitting. It continues to improve on the training data while its performance on unseen data worsens.

This pattern implies a lack of generalization and highlights the importance of applying stronger regularization techniques or introducing early stopping. It may also suggest that the model is too complex for the size or diversity of the dataset used.

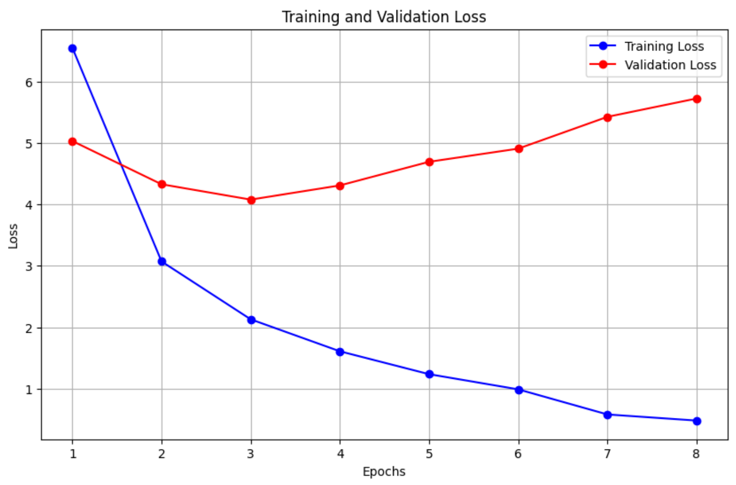

Fig. 6. Learning curves for the BiDAF model showing training and validation loss.

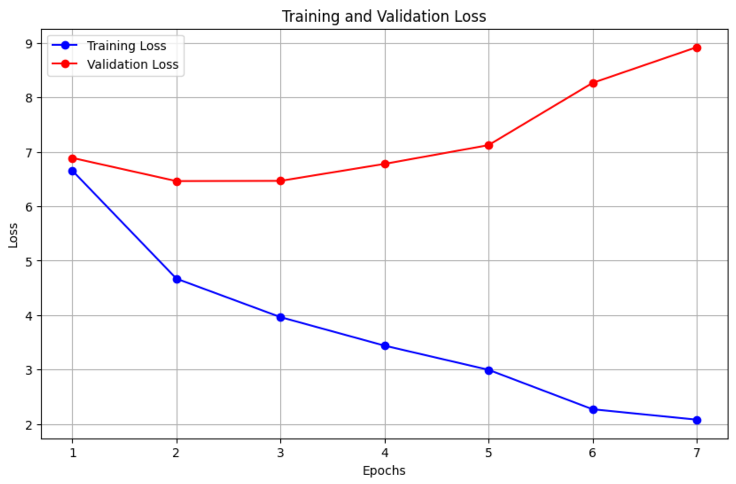

Fig. 7. Learning curves for the DrQA model showing training and validation loss.

DrQA Figure 7 shows the training and validation loss across 7 epochs. The training loss decreases consistently, indicating that the model is learning to minimize error on the training set.

However, the validation loss starts to diverge significantly from the training loss after the third epoch. It rises steadily and sharply in the later epochs, suggesting strong overfitting , the model is adapting too well to the training data and failing to generalize to unseen data.

This pattern points to issues such as an overly complex model, inadequate regularization, or a training set that does not represent the validation set well. The rapid rise in validation loss highlights the need for earlier stopping, dropout, or possibly simplifying the model architecture to prevent overfitting.

7 Discussion

The results highlight a significant disparity in performance across the different model architectures we experimented with. The BiDAF model stands out as the most effective, achieving the highest scores on both the joint exact match and span F1 metrics on the validation set. This suggests that the bi-directional attention mechanism is particularly well-suited for capturing the nuanced interactions between the question and the context required for accurate span prediction. The DrQA-inspired model also shows reasonable performance, indicating the value of combining BiLSTMs with an attention mechanism.

In contrast, the models that process the question and context more independently or rely solely on a Transformer encoder with GloVe embeddings struggled to achieve comparable results. The poor performance of the Dual-RNN model underscores the limitation of summarizing the entire context and question into fixed-size vectors without a more fine-grained interaction mechanism. Similarly, the weaker results from the GloVe Transformer and the QA Separated Transformer suggest that without more sophisticated architectural adaptations or pre-training on larger datasets, these models may not effectively learn the complex relationships needed for precise answer extraction in our limited data setting.

It is also worth noting the differences between training and validation performance. While some models achieved high scores on the training data, their performance dropped on the validation set, indicating potential overfitting. The BiDAF model showed a relatively smaller gap between training and validation scores, suggesting better generalization capabilities.

8 Limitations

Our project faced several limitations that likely impacted the performance of our models:

- **Limited Dataset Size:** Using only 20,000 answerable examples from the SQuAD 2.0 dataset is a significant reduction from the full dataset. This smaller size likely limited the ability of more complex models, especially the transformer-based ones, to learn robust representations and generalize well.

– **Restricted Model Complexity:** The constraint of using only three layers (excluding input and output) limited the depth and capacity of our models. Deeper networks, particularly in transformer architectures, often benefit from their ability to learn hierarchical features.
– **Computational Resources:** Limited access to high-end GPUs restricted the batch sizes, training times, and the ability to experiment with larger or more computationally intensive models. This also constrained our ability to perform extensive hyperparameter tuning.
– **Model Architecture Constraints:** The instruction to use either RNN-based or attention-based models, but not a combination (initially), prevented us from fully exploring architectures that the literature suggests are highly effective, such as those combining RNNs with attention mechanisms. While we did implement BiDAF which does this, our exploration of other hybrid models was limited.
– **Data Filtering:** Filtering the dataset based on context length, while helpful for managing computational resources, might have introduced biases or excluded valuable examples that could have improved model generalization.
– **Tokenization and Vocabulary Size:** Our choice of a 10,000 subword vocabulary using BPE might not have been optimal. Exploring different tokenization strategies or vocabulary sizes could potentially impact performance.
– **Lack of Pre-training:** Except for the GloVe embeddings used in one of the transformer models, our models were trained from scratch on the limited SQuAD subset. Leveraging large pre-trained language models like BERT, even with fine-tuning on our smaller dataset, could have significantly boosted performance.

These limitations highlight areas for potential improvement in future work, such as utilizing the full SQuAD dataset, experimenting with deeper and more complex architectures (if computational resources allow), leveraging pre-trained models, and exploring different data preprocessing and tokenization techniques.

Bibliography

[1] Chen, D., Fisch, A., Weston, J., and Bordes, A. (2017). Reading wikipedia to answer open-domain questions.

[2] Seo, M., Kembhavi, A., Farhadi, A., and Hajishirzi, H. (2018). Bidirectional attention flow for machine comprehension.