Transformer-based Models for Answer Extraction in Text-based Question/Answering

by

Marzieh Ahmadi Najafabadi

A thesis submitted to the
School of Graduate and Postdoctoral Studies in partial
fulfillment of the requirements for the degree of

Master of Science (MSc) in Computer Science

Faculty of Science
University of Ontario Institute of Technology (Ontario Tech University)

Oshawa, Ontario, Canada

April 2023

© Marzieh Ahmadi Najafabadi, 2023

THESIS EXAMINATION INFORMATION

Submitted by: **Marzieh Ahmadi Najafabadi**

Master of Science in Computer Science

Thesis Title: Transformer-based Models for Answer Extraction in Text-based Question/Answering

An oral defense of this thesis took place on April 11th, 2023 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Dr. Andrew Houge
Research Supervisor	Dr. Heidar Davoudi
Examining Committee Member	Dr. Mehran Ebrahimi
Thesis Examiner	Dr. Amirali Salehi-Abari

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

Abstract

The success of transformer-based language models has led to a surge of research in various natural language processing tasks, among which extractive question-answering/answer span detection, has received considerable attention in recent years. However, to date, no comprehensive studies have been conducted to compare and examine the performance of different transformer-based language models in the task of question-answering (QA). Furthermore, while these models can capture significant semantic and syntactic knowledge of a natural language, their potential for enhancing performance, in QA, through the incorporation of linguistic features remains unexplored. In this study, we compare the efficacy of multiple transformer-based models for the task of QA, as well as their performance on particular question types. Moreover, we investigate whether augmenting a set of linguistic features extracted from the question and context passage can enhance the performance of transformer-based language models in QA. In particular, we examine a few feature-augmented transformer-based architectures for the task of QA to explore the impact of these linguistic features on several transformer-based language models. Furthermore, an ablation study is conducted to analyze the individual effect of each feature. Through conducting extensive experiments on two question-answering datasets (i.e., SQuAD and NLQuAD), we show that the proposed framework can improve the performance of transformer-based models.

Keywords: Question-Answering; Answer Span Detection; Transformer-based Models; Pre-trained Models

Author's Declaration

I hereby declare that this submission is entirely my own work, in my own words, and that all sources used in researching it are fully acknowledged. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize the University of Ontario Institute of Technology (Ontario Tech University) to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize University of Ontario Institute of Technology (Ontario Tech University) to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

_____ Marzieh Ahmadi Najafabadi

Statement of Contributions

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication. I have used standard referencing practices to acknowledge ideas, research techniques, or other materials that belong to others. Furthermore, I hereby certify that I am the sole source of the creative works and inventive knowledge described in this thesis.

Acknowledgements

I would like to express my sincere appreciation to my supervisor Dr. Kourosh Davouid for his patience, help, and support that allowed me to successfully accomplish my research. He trusted me and gave me the opportunity to be a member of his team.

I would like to express my gratitude and appreciation to the faculty and the Computer Science program members.

Last but not least, I would love to give my special regards to my amazing family and express my deepest gratitude to my beloved sister, Malihe Ahmadi, for her endless support. This journey would not have been possible without their patience, love and support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Extractive question-answering, which aims to find the answer to a question within the given passage, has became critical among natural language processing tasks recently. Transformer-based [91] language models are a type of neural network architecture that have been used in question-answering tasks. The transformer [91] architecture is designed to address the limitations of traditional neural network architectures, which struggle with handling long sequences of text due to the vanishing gradient problem. In a question-answering task, considered in this thesis, the inputs to the model are a question and a passage of text that may contain the answer to the question. Transformer-based language models process the input text using a series of self-attention mechanisms to identify important features and relationships between the words in the text. Then, they use this information to generate an answer (i.e. span of words within the passage) to the question.

Generally speaking, a question can be either factoid or non-factoid. A factoid question is asking about an entity or an event (e.g. person, organization, location, etc.), which is a simple and short fact. However, a non-factoid question is asking about a body of information and the answer to these questions can be very complex, containing

definitions, opinions, examples, etc. Non-factoid questions can be defined as open-ended questions as well, which cannot be answered by a simple *yes* or *no* or any other *static* response [28]. Thus, the answer to a non-factoid question can be extended to multiple sentences or paragraphs [82]. For example, *What* and *Who* questions are factoid questions, while *How* and *Why* questions are considered as non-factoid questions. When we ask *"How to cook burgers?"* we're looking for detailed description and explanation. Despite when we ask *"What is a female rabbit called?"* we're asking about a simple fact. Non-factoid questions are closer to real-life questions and they play a crucial role in the quality of question-answering [28].

Automatic question-answering (in general) can be helpful in finding information efficiently. In customer support use cases, common questions get asked frequently. Question-answering enables us to create a chat bot from existing supporting content to handle customer queries. Moreover, with the help of question-answering models we can augment search results with instant answers in order to provide the users with immediate access to relevant information to their queries. Thus, question-answering models are useful in most of the information retrieval use cases.

1.2 Problem Definition

In this thesis, we consider utilizing transformer-based models for the task of extractive question-answering, where given a question and a passage, the goal is to identify the span of words within the passage which contain the answer to the question. To elaborate, the extractive question answering task involves a question-passage pair as input, with the desired outputs being the start and end positions within the provided passage that accurately encapsulate the answer to the given question.

Question-answering and information retrieval are two distinct yet interconnected research domains. While extractive question-answering aims to provide precise and accu-

rate answers to a given question, information retrieval systems, on the other hand, do not proffer explicit answers to questions, but rather present relevant documents or parts of documents, requiring the user to extract relevant information themselves.

Transformer-based language models introduce promising performance in most of the natural language processing tasks by proposing the attention mechanism. However, to date, they are not well studied in the field of extractive question-answering. Moreover, the performance of transformer-based models on different question types (e.g. what, how, why, when, where, etc.) haven't been explored yet.

This research aims to compare the performance of different state-of-the-art transformer-based models in the extractive question-answering task. We also investigate how incorporating a set of off-the-shelf linguistic features can enhance the performance of these transformer-based models. In particular, this study seeks to explore whether the augmentation of transformer-based models' architecture with these linguistic features can improve their overall performance. Finally, we investigate the performance of different transformer-based language models in handling long context passages in the extractive question-answering task.

The SQuAD [73] dataset, known for its high quality, is selected as the benchmark for evaluating both the proposed and investigated models by this thesis in the extractive question-answering task. Each instance in the dataset consists of a triad comprising a question, passage, and ground truth answer. For each question-passage pair, we evaluated the predicted answer based on the available ground truth answer for that pair. Further explanation on data collection and generation in SQuAD [73] dataset is given in Chapter 5.

1.3 Contribution

We summarize the main contributions in the following.

- This thesis studies and compares different transformer-based [91] models, such as BERT [19], RoBERTa [52], BART [50] and Longformer [5] in the task of question-answering.

- We explore and compare the performance of different transformer-based models on different question *types*.

- We proposed three variants of a feature-augmented architecture, which augments the transformer-based models with a few different token-level linguistic features extracted from both question and context passage, to explore if linguistic features can improve the performance of transformer-based models.

- We aim to conduct a comprehensive investigation of individual features and their impact on various transformer-based models through an ablation study.

1.4 Thesis Outline

This thesis is organized in six Chapters and structured as follows:

- Chapter 2 introduces a few state-of-the-art language models that are aimed to understand text for different natural language processing tasks. These models achieved state-of-the-art performances on most of language modeling tasks as they have a very precise understanding of language and are all based on transformers [91] architecture.

- Chapter 3 focuses on the task of question-answering and presents different settings and directions of this field. Finally, two distinct question-answering methods are explained in depth to provide a clearer picture of the diverse settings and models that are available in this field of research.

- Chapter 4 presents our proposed transformer-based feature-augmented architecture designed for the task of question-answering, which predicts the answer span to a given question from the provided passage. In this Chapter, we also describe the question-answering related details of backbone models used in our design, which are a few transformer-based models.

- Chapter 5 reports the results from different variants of proposed feature-augmented architecture. In this chapter, we further compare the performance of various transformer-based feature-augmented models for each specific question types on SQuAD [73]. Additionally, we conduct an ablation study through which we evaluate the effect of each feature on the performance of the feature-augmented architecture. Lastly, this chapter presents case studies, chosen from SQuAD [73], that demonstrate the influence of features on the backbone model.

- In Chapter 6, we highlight the key contributions and some of the limitations of the thesis research. The Chapter also presents some interesting future directions along which the thesis research can be extended.

1.5 Software & Source Code

Software

The implementation of the transformer-based language models for the task of question-answering as well as the proposed feature augmentation architecture, along with its variants, and all of the different experiments that we conducted are written in Python programming language. Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Python is meant to be an easily readable language. Majority of the training and experiments in this research, were carried out on a cluster of 4 NVIDIA

V100 GPUs (Graphical Processing Units), while CPUs (Central Processing Units) were used for data loading and pre-processing purposes. Following packages are examples of Python packages used in this thesis.

- **Transformers** provides APIs and tools to easily download and train state-of-the-art pre-trained models. Using pre-trained models can reduce the compute costs and save the time and resources, which are required to train a model from scratch. These models support common tasks in different modalities, such as: Natural Language Processing (NLP), Computer Vision, Audio, etc.
  https://huggingface.co/docs/transformers

- **PyTorch** is an open source machine learning library used for developing and training computer vision and Natural Language Processing (NLP) tasks. This package provides two high-level features:

  - Tensor computation (like NumPy) with strong GPU acceleration.
  - Deep neural networks built on a tape-based autograd system.

  https://pytorch.org/

- **SpaCy** is an open-source software library for Natural Language Processing (NLP) in Python. It is designed to be fast and efficient, providing advanced NLP capabilities such as tokenization, part-of-speech (POS) tagging, named entity recognition (NER), dependency parsing, and sentiment analysis. SpaCy also supports multiple languages and integrates well with other popular libraries in the NLP and machine learning communities, making it a popular choice for NLP-related tasks and projects.
  https://spacy.io/

Source Code

You can access the Python implementations of our models, evaluation metrics, and all the experiments at the following link.

https://github.com/MarziehAhmadiNa/Transformer-based-Models-For-Answer-Extraction

Chapter 2

Background

Since the focus of this thesis is on transformer-based language models, which are natural language models that are trained on unsupervised task and can be fine-tuned for different natural language processing downstream tasks, for the task of question-answering by comparing and improving their performance, in the following sections we will give a brief introduction to each of the studied transformer-based language models by this work. Prior to delving into the specifics of each model, we will provide an introduction to transformers [91], which is the fundamental architecture underlying all of the models under investigation by this thesis.

2.1 Transformers

Transformers, which were introduced in the paper *Attention is All You Need* [91], were developed as an alternative to Recurrent Neural Networks (RNNs), Long Short-term Memories (LSTMs) [34], and Gated Recurrent Neural Networks (GRNNs) [15]. These traditional models had previously achieved state-of-the-art performance in sequence modeling [85] tasks and transduction problems (also known as sequence labeling tasks), such as language modeling and machine translation [4, 14]. Further research has also explored the effectiveness of recurrent language models in an encoder-decoder format [42, 55, 98].

When translating a sentence from English to French using a recurrent model, the network would take an English sentence as input and process the words one by one, sequentially generating the corresponding French words. The order of the words in a sentence is critical in language, and rearranging them can significantly alter the meaning of the sentence. Thus, the key factor here is the sequence. For example, the sentence *"Jane went looking for trouble"* has a vastly different meaning from *"Trouble went looking for Jane"*. In other words, any language model must account for word order, and RNNs do this by examining each word in sequence, one at a time.

Despite their usefulness, recurrent models faced several issues. Firstly, they struggled to handle large sequences of text, such as long paragraphs or essays, as they tended to forget the beginning by the time they analyzed the end. On top of that, RNNs were difficult to train due to their sequential word processing, making parallelization with multiple GPUs impossible. As the sequence lengths become longer, the memory constraints restrict the ability to batch across examples, and this becomes increasingly critical. While recent studies have made considerable improvements in computational efficiency by employing factorization tricks [48] and conditional computation [80], the fundamental constraint of sequential computation persists. Moreover, a slow training model can not be trained on much data. This is where transformers revolutionized the field.

Initially, transformers were created for translation purposes, but their application quickly expanded to various natural language processing tasks. Unlike recurrent models, transformers could be parallelized efficiently, enabling the training of large models with sufficient resources. The success of transformers can be attributed to three key innovations: positional encoding, attention, and self-attention, which will be explained separately in the following.

The concept of positional encoding involves assigning a numerical value to each word in a sentence based on its position in the sequence before feeding it into a neural network.

This method captures information about the order of the words within the data itself, rather than relying on the structure of the network. As the neural network is trained on multiple text data, it learns to interpret these positional encodings and understand the significance of word order. This is particularly important in models lacking recurrence or convolution, which require information about the relative or absolute position of tokens in a sequence. However, assigning numbers based on word order may not be the optimal approach as it can lead to large values and difficulty with longer sentences not seen in the training data. There exists a wide range of positional encoding options, including both learned and fixed methods [25]. In transformers, a sinusoidal function is chosen for positional encoding. This innovation has contributed to the success of transformers by simplifying the training process compared to recurrent models.

Attention mechanisms have become a crucial component of effective sequence modeling and transduction models for various tasks. They enable the modeling of dependencies between input and output sequences regardless of their distance, as demonstrated by their application in [4, 45]. This mechanism is a neural network structure that enables a language model to consider every word in the input sentence when making decisions related to the task at hand, such as translating a word in the input sequence. Unlike a suboptimal approach where each word is translated individually and in sequence, the attention mechanism can account for the fact that words can be reordered when translating from one language to another. The model is capable of looking at different words during translation, and this is learned from data over time. By training on thousands of examples of French and English sentence pairs, the model acquires knowledge of grammatical rules, such as gender, word order, and plurality, required for accurate sentence translation.

With attention mechanism being already invented before this paper, the real innovation in transformers was self-attention (also called intra-attention), which was a twist on traditional attention. Self-attention is an attention mechanism that enables the calculation of a sequence representation by relating various positions within a single sequence.

This approach has proven to be effective in multiple tasks such as reading comprehension [12], abstractive summarization [65], textual entailment [64], and generating task-independent sentence representations [51].

The self-attention mechanism enables the model to comprehend the underlying semantics in language and construct a network capable of solving various language tasks. Transformers examine a wide range of textual data, allowing them to automatically develop an internal representation of language. The quality of this internal language representation learned by the neural network is directly proportional to its ability to perform any language task proficiently. Incorporating attention onto the input text itself can be an extremely effective approach to assist the neural network in understanding language. For instance, the term ***server*** in the sentences *"Server, can I have the check?"* and *"Looks like I just crashed the server!"* has two completely distinct meanings that we can only discern by analyzing the context of the surrounding words. Self-attention enables the neural network to comprehend a word within the context of the words adjacent to it. Thus, when processing the term ***server*** in the first sentence, the model may focus on the term ***check*** to differentiate it from a human server versus a mail server. However, in the second sentence, the model may attend to the word ***crash*** to deduce that the server is a machine. Self-attention can help the network not only disambiguate words, but also recognize word tense and parts of speech. Rather than merely looking at preceding hidden vectors when considering a word embedding, self-attention combines all other word embeddings (including those that appear later in the sequence) in a weighted manner.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.1}$$

Attention refers to a function that takes a set of queries with their respective keys and values as input and produces a set of outputs. Transformers use a specific type of attention called *Scaled Dot-Product Attention*, where the function takes in queries Q and keys K with a dimension of d_k, and values V with a dimension of d_v. The function first

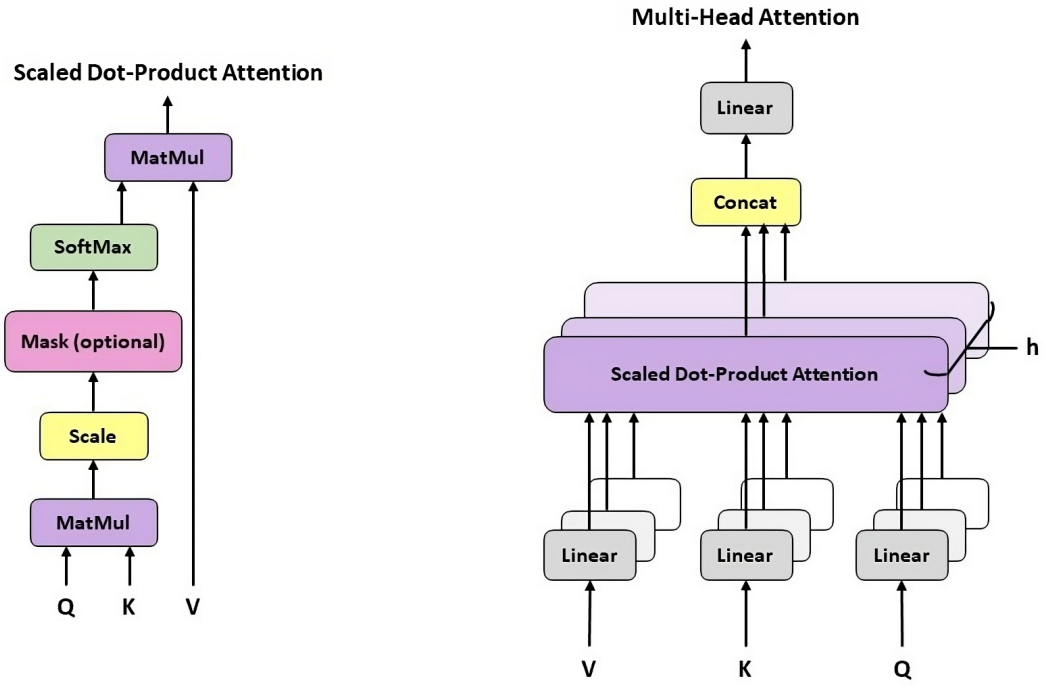


Figure 2.1: Scaled Dot-Product Attention vs. Multi-Head Attention. Multi-Head Attention consists of multiple parallel attention layers.

calculates the matrix multiplication between Q and K^T (K^T is the transpose of vector K), scales the product by dividing it by $\sqrt{d_k}$ to improve the gradient flow, and then applies a softmax function. Scaling is specially important in cases when the value of matrix multiplication is too big. Finally, to preserve the most critical input values, the function computes the matrix multiplication between these softmax scores and values vector V. In other words, the function keeps the words with high softmax probability. The formula for computing the scaled dot-product attention matrix is presented in equation 2.1 [91].

In transformer architecture, self-attention is computed independently, and it is performed h times in parallel using different learned linear projections. This method is referred to as *Multi-Head Attention*. The outputs of multiple self-attention operations are concatenated and then linearly transformed. Equation 2.2 contains the formula for multi-head attention computation, in which the projections (W^O, W_i^Q, W_i^K and W_i^V) are

parameter matrices. To elaborate, W_i^Q is the parameter matrix for linear projection of Q in $head_i$, W_i^K is the parameter matrix for linear projection of K in $head_i$ and W_i^V is the parameter matrix for linear projection of V in $head_i$. Ultimately, W^O is the parameter matrix for linear projection after concatenation of multiple attention heads [91].

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{2.2}$$

$$\text{where } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{2.3}$$

As mentioned previously, transformers are designed to address multiple language tasks in a sequential manner while also handling long-range dependencies between words in a sentence. Following an encoder-decoder architecture similar to other advanced neural models for sequence transduction [4, 14, 85], the transformer model employs various components as illustrated in Figure 2.2 [91].

Specifically, the encoder encodes the input sequence into a fixed-length vector known as the context vector, associating the input tokens together while utilizing the self-attention mechanism to learn their representation. Similarly, the decoder employs self-attention when processing the encoder's output while iterating over the generated output sequence, with information drawn from the context vector. The encoder and decoder are composed of a stack of $N = 6$ identical layers, which each layer comprise two and three sub-layers, respectively [91].

The encoder consists of two sub-layers, namely the multi-head self-attention mechanism and a position-wise fully connected feed-forward network. To ensure better performance, each of these sub-layers is augmented with a residual connection [30] and followed by layer normalization [3]. Consequently, the output of each sub-layer is computed as $LayerNorm(x + SubLayer(x))$, where $SubLayer(x)$ represents the function of the corresponding sub-layer [91].

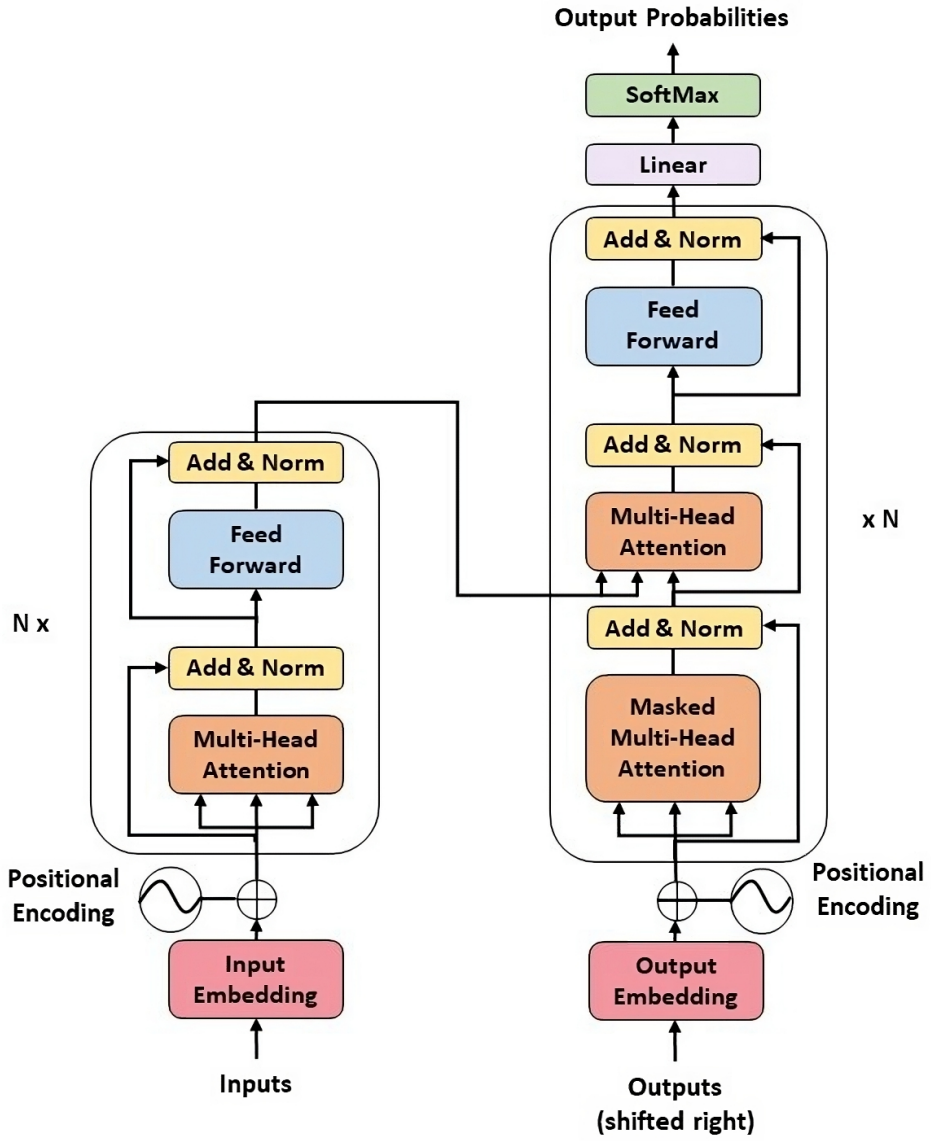


Figure 2.2: Transformer architecture. The left side of this figure shows the stack of encoders, while the right side is showing the stack of decoders.

The decoder includes a third sub-layer that conducts multi-head attention on the encoder stack's output, along with the two sub-layers in each encoder layer. Like the encoder, each sub-layer is surrounded by residual connections [30] and followed by layer normalization [3]. The decoder's self-attention sub-layer is also adjusted to *Masked Multi-Head Attention*, to prevent positions from attending to future positions. This masking, along with the fact that the output embeddings are shifted by one position, guarantees

that the predictions for position i are solely reliant on the known outputs that come before position i.

To summarize, transformers offer several benefits over recurrent models. Transformers are capable of comprehending the relationship between sequential elements that are widely spaced apart, leading to greater accuracy. Additionally, transformers assign equal attention to all elements in a sequence. Furthermore, transformers are capable of managing and training larger datasets in a shorter period of time.

2.2 BERT

BERT [19], short for Bidirectional Encoder Representation from Transformers, is a language tool that surpasses other existing models in language understanding and learning. It achieves this by utilizing an innovative approach to applying bidirectional training of the transformer [91] to language modeling.

The effectiveness of language model pre-training has been demonstrated in enhancing numerous natural language processing tasks [17, 35, 67, 70]. Moreover, modern NLP systems rely heavily on pre-trained word embeddings, which have been shown to provide substantial benefits compared to embeddings that are learned from scratch [89]. Some of the tasks that benefit from language model pre-training encompass sentence-level problems like natural language inference [7,96] and paraphrasing [20], which seek to assess the correlation between sentences by examining them as a whole, and token-level operations such as named entity recognition [78] and question-answering [73], where models need to generate detailed results at the token level.

There are two primary methods for utilizing pre-trained language models in downstream tasks: feature-based and fine-tuning. The feature-based technique, exemplified by ELMo [67], utilizes task-specific architectures that incorporate the pre-trained representations as additional features. On the other hand, the fine-tuning method, such as

the Generative Pre-trained Transformer (OpenAI GPT) [70], incorporates minimal task-specific parameters and is trained on downstream tasks by fine-tuning all pre-trained parameters. During pre-training, both approaches share the same objective function, in which they utilize unidirectional language models to acquire general language representations.

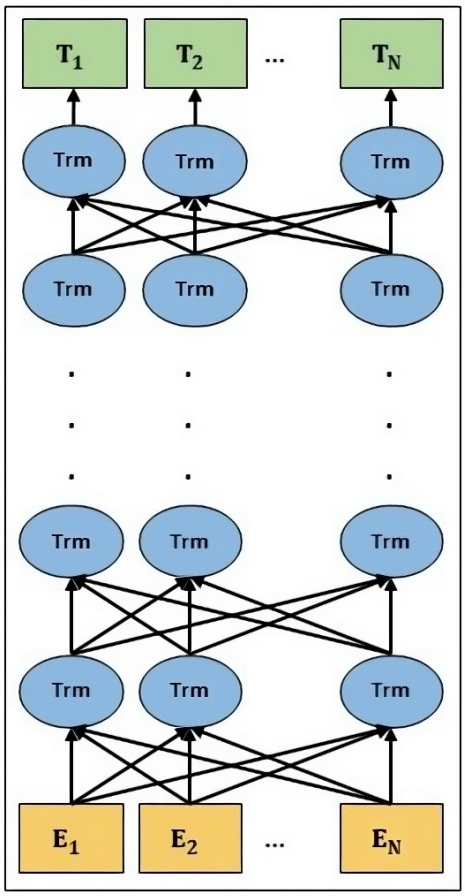


Figure 2.3: BERT uses a bidirectional transformer encoder, in which the representations are jointly conditioned on both left and right context in all the layers.

Unidirectional language models process input sequences in a left-to-right or right-to-left manner. Enforcing these limitations is not optimal for sentence-level tasks and can be highly detrimental when implementing fine-tuning based techniques for token-level tasks like question-answering, which require incorporating context from both directions. BERT is considered to be bidirectional that trains the language model bidirectionally, which

enables it to have a more profound and comprehensive understanding of the language context and flow compared to a single directional model. In essence, a bidirectional model can comprehend the meaning of a word based on its entire context. Figure 2.3 depicts BERT model's bidirectional architecture [19].

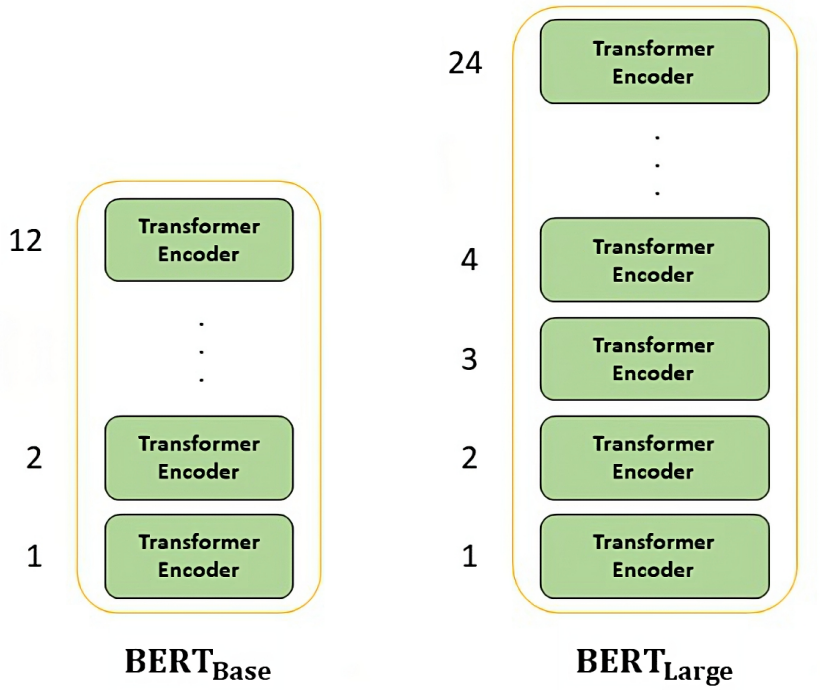


Figure 2.4: Two versions of BERT, depending on the number of encoders stacked over top of each other. BERT-Base model (left) has 12 Encoder Layers, with 768 Hidden Size and 12 Self-Attention Heads ($L = 12$, $H = 768$, $A = 12$). In contrast, BERT-Large model (right) has 24 encoder layers, with 1024 Hidden Size and 16 Self-Attention Heads ($L = 24$, $H = 1024$, $A = 16$).

BERT consists of two phases: pre-training and fine-tuning. In essence, BERT is pre-trained on unlabeled text datasets (the BooksCorpus [107] with 800M words and English Wikipedia with 2,500M words) to help the model understand language, followed by fine-tuning on specific tasks. During the fine-tuning phase, the model is initialized with the pre-trained parameters, and then all parameters are fine-tuned using task-specific data.

BERT is applicable to various downstream tasks, including Sentiment Analysis, Question-Answering, Text Classification, among others. It is noteworthy that BERT's architecture remains consistent across various tasks, and there are only minor distinctions between the pre-trained and ultimate downstream architecture. Each downstream task has its own separate fine-tuned model, while they are all initialized with the same pre-trained parameter [19].

The goal of pre-training is to teach BERT *"What is Language?"* and *"What is Context?"*. To achieve this, BERT is trained simultaneously on two unsupervised tasks [19]:

- **Masked Language Modeling (MLM)** inspired by the Cloze task [87], for which BERT receives a sentence in which some words are randomly masked, and the objective is to independently and simultaneously predict the original token for these masked tokens. Unlike pre-training with a left-to-right language model, the MLM objective allows the representation to combine the left and right contexts, enabling pre-training of a deep bidirectional Transformer [19].

- **Next Sentence Prediction (NSP)**, in which BERT is presented with two sentences, and it determines whether the second sentence is the subsequent sentence in the original document, aiding BERT's comprehension of context across multiple sentences [19].

By jointly training both unsupervised tasks and minimizing their combined loss, BERT develops a robust understanding of language [19].

In order to enable BERT to tackle a range of downstream tasks, the input representation can distinctly represent a single sentence or a pair of sentences (such as a question and answer) using a single token sequence [19].

During the random masking procedure for Masked Language Model, a randomly selected word is masked 80% of the time (substituted with the [MASK] token). For instance, `the man went to the store` →`the man [MASK] to the store`. While in

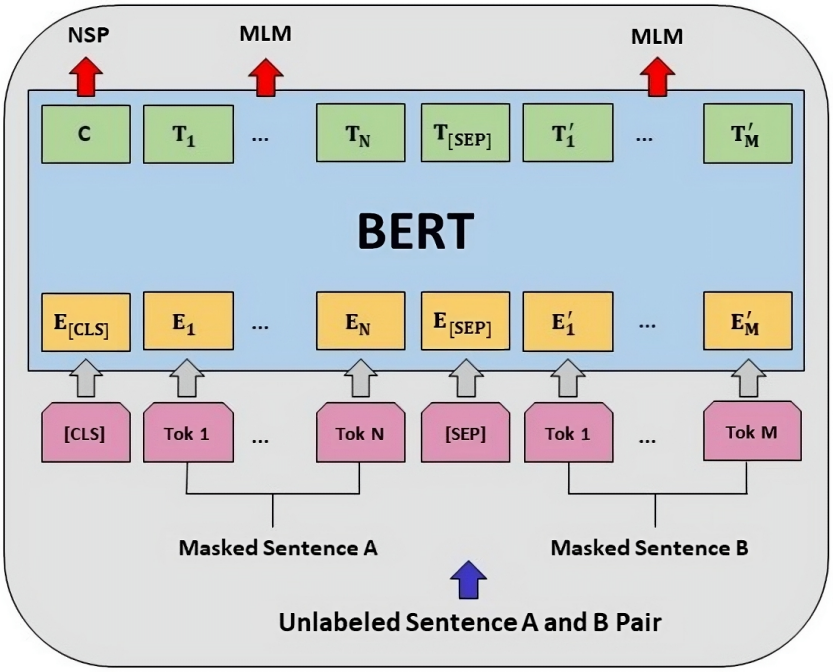


Figure 2.5: BERT overall pre-training procedure. [CLS] is a special token, which is added to the start of every input sequence and [SEP] is another special token, which is used to separate first sentence from the second sentence.

10% of cases, this randomly chosen word is replaced with another random word, e.g. `the man went to the store` →`the man spoke to the store`. The remaining 10% of the time, the randomly chosen word remains unchanged. For example, `the man went to the store` →`the man went to the store`. This procedure forces the transformer to preserve a contextual representation of every single input tokens, since it cannot determine which token has been replaced with a random token. [19].

For the next sentence prediction task, 50% of the time the inputs consist of a pair of sentences where the second sentence is the immediate subsequent sentence. While in the other 50% of cases, a sentence is randomly selected from the corpus to serve as the second sentence (assuming it will be unrelated to the first sentence). This can be

illustrated using an example:

Input = `[CLS] the boys went to [MASK] store [SEP] they bought different kinds [MASK] candies [SEP]`

Output = `IsNext`

Input = `[CLS] the boys [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP]`

Output = `NotNext`

The NSP objective of BERT shares similarities with the representation learning objectives employed in [38] and [53]. However, these previous works only transferred sentence embeddings to downstream tasks, while BERT transfers all parameters to initialize the downstream task's model parameter [19].

During the pre-training phase (as depicted in Figure 2.5), the input consists of a pair of sentences (sentence A followed by sentence B) with certain words being masked. Each token corresponds to a word and is transformed into initial embeddings by the model. These embeddings are produced by combining three vectors: Token Embeddings, Segment Embeddings, and Position Embeddings. The Token Embeddings are pre-trained embeddings, where this paper utilizes *WordPieces* [98], which has a vocabulary of 30,000 tokens. The Segment Embedding is a vector that encodes the sentence number and specifies which sentence each token belongs to (either sentence A or B). The Positional Embedding is a vector that encodes the word's position within that sentence. The Segment and Positional Embeddings are crucial for preserving temporal ordering, as all vectors are fed into BERT simultaneously, and language models require this ordering to be maintained. Figure 2.6 presents an example to illustrate how these embeddings are constructed.

The binary output for the next sentence prediction is represented by C in the output sequence. C will output 1 if sentence B follows sentence A in context and 0 otherwise. Furthermore, all word vectors T_i in the output sequence correspond to the outputs for the

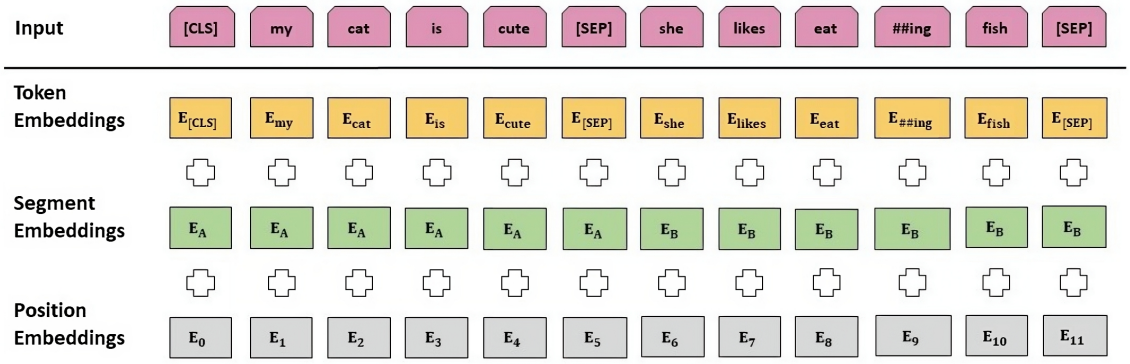


Figure 2.6: An example of BERT initial embeddings calculation.

masked language model problem. All of the word vector T_i have the same size and are generated concurrently. During training, the goal is to minimize the loss, which involves passing each word vector into a fully connected layered output with the same number of neurons as the vocabulary size. Then a softmax activation function is applied to this output layer, consisting of 30,000 neurons in this case, in order to convert a word vector into a distribution. The distribution's actual label is a one-hot encoded vector for the actual word, allowing for comparison with the predicted distribution. The network is then trained using the *Cross Entropy Loss*. Note that the output contains all words, regardless of whether they were masked or not. However, the loss only considers the prediction of the masked tokens and it disregards all the other tokens that are output by the network. By doing so, the model ensures that the focus is on predicting these masked values, increasing the model's context awareness [19].

The aim of fine-tuning BERT is to enable it to effectively perform specific NLP tasks. For instance, in a question-answering task, we need to substitute the fully connected output layers of the network with a linear layer followed by a softmax, that can provide the answer span to the given question. Following this, we can perform a supervised training using a question-answering dataset. The training process will not take much time as only the output parameters are being learned from scratch, while the remaining

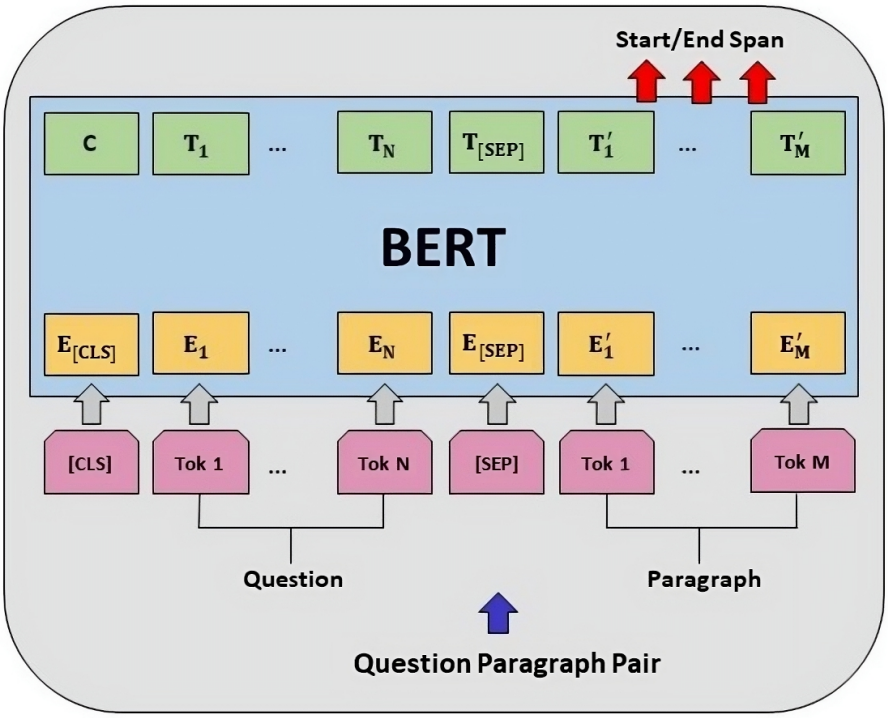


Figure 2.7: BERT fine-tuning procedure for question-answering task. Despite different output layers, the same structure is used for both pre-training and fine-tuning.

model parameters are just fine-tuned slightly.

During the fine-tuning phase, the model is trained by adjusting both the input and output layers. In the case of question-answering task for example (as shown in Figure 2.7), the input consists of the question followed by a passage containing the answer. In the output layer, the model would generate the start and end indexes that encapsulate the answer.

2.3 RoBERTa

RoBERTa [52], which stands for Robustly optimized BERT pre-training approach, is a modified version of BERT [19] that trains the model for longer time, with a larger batch over more data (different datasets) and longer sequences of data. This model re-

implements BERT and fixes some hyper parameters while tuning the others [52]. The re-built model pre-training objective is in competition with other training objectives that have been proposed before this work, such as perturbed auto-regressive language modeling as described by XLNet [103].

RoBERTa uses the same architecture as BERT but removes the next sentence prediction pre-training objective. As mentioned before, next sentence prediction objective is necessary to incorporate long distance relationships within the corpus. This pre-training task can be helpful in some of the downstream tasks such as Natural Language Inference, in which understanding the relationship between pairs of sentences is required and necessary. However, by conducting a few experiments and defining different formats of inputs, the authors of this paper concluded that training without the next sentence prediction loss outperforms the original BERT model's performance in most of the downstream tasks. Thus, eliminating the next sentence prediction loss can improve the performance of the model [52].

Furthermore, the authors conducted a comparison between static masking and dynamic masking strategies for the masked language modeling task. The original BERT model employed static masking, where the random masking is done only once during pre-processing. Moreover, in static masking, to prevent the model from being trained on the same masked sequence in every epoch, each training instance was masked in ten different ways throughout the training. However, dynamic masking generates a new masked sequence for each epoch during training rather than pre-computing and saving the masking pattern. The authors found that using dynamic masking resulted in a slight improvement in the model's performance compared to using static masking [52].

To summarize, RoBERTa is trained with larger mini batches on more data using dynamic masked language modeling, excluding the next sentence prediction loss for extended period of time [52].

2.4 BART

BART [50], which stands for Bidirectional Auto-Regressive Transformers, is a transformer-based model which can be employed for a wide range of text generation tasks, including translation and summarization, etc. as well as various other natural language processing tasks like question-answering, etc. In essence, BART is a sequence-to-sequence architecture for natural language comprehension, generation, and translation that incorporates denoising capabilities [50]. BART utilizes the standard sequence-to-sequence transformer [91] architecture, with one exception: ReLU activation functions are replaced with GeLUs [31].

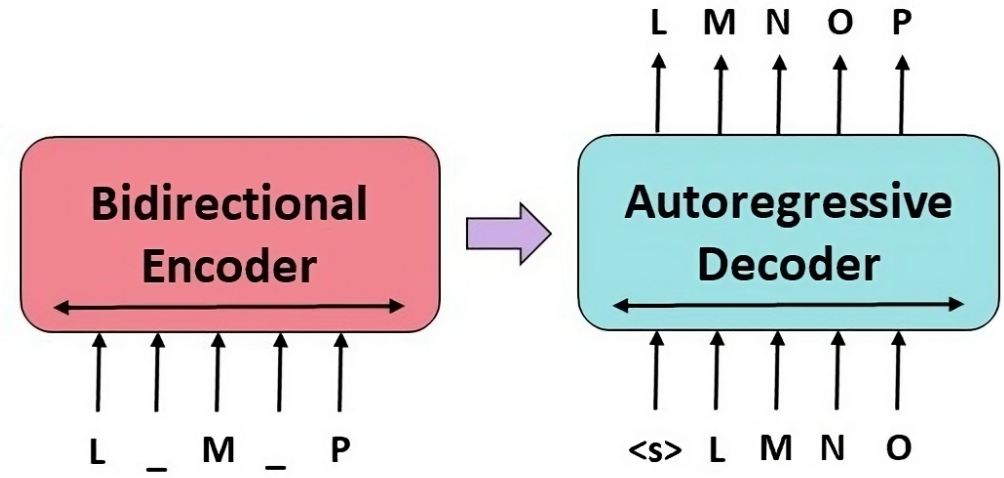


Figure 2.8: BART model: Denoising sequence to sequence auto-encoder. The corrupted document, on the left, is encoded bidirectionally using the transformers encoder. Then the likelihood of the original document, on the right, is computed through an auto-regressive decoder.

The architecture of BART, depicted in Figure 2.8, is that of an auto-encoder that aims to remove noise from a corrupted document to recover the original version. The model consists of a bidirectional transformer encoder that takes in the noisy text, as well as a left-to-right transformer auto-regressive decoder. To ensure effective noise transformation,

the input to the encoder should not align with the decoder's output. During the pre-training phase, the model optimizes the negative log likelihood of the original text [50].

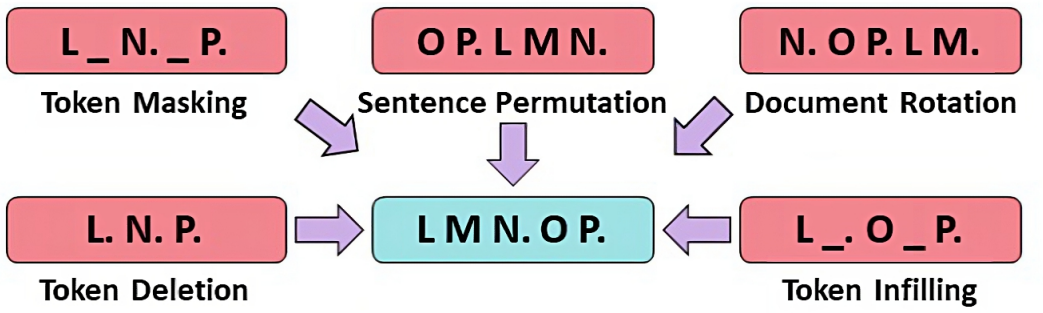


Figure 2.9: Different transformation functions BART uses in pre-training step for corrupting the original input text.

In the pre-training step, BART formulates various innovative transformation objectives (Figure 2.9):

- **Token Masking:** Similar to the masking strategy proposed by BERT [19], random tokens are masked (replaced by [MASK] or _) in the input sequence, enabling prediction of the masked tokens based on their contextual meaning [50].
- **Token Deletion:** Random tokens within the input sequence are deleted. The model is tasked with identifying the positions of the missing tokens and predicting their values. Thus, this task is more complicated compared to the token masking task [50].
- **Text Infilling:** Instead of masking only one token, a span of tokens is replaced with a single [MASK] token (inspired by SpanBERT [40]), with the length of the span ranging from 0 to 3 tokens. Thus, the model learns to identify the number of missing tokens and predict their values [50].
- **Sentence Permutation:** The document's sentences, which are segmented based on full stops, are randomly shuffled [50].

- **Document Rotation:** A token is randomly selected from the document and is moved to the beginning of the document, with the purpose of teaching the model to identify the correct start for the document [50].

The model is presented with various transformation functions to learn different aspects of language and improve its ability to adapt and generalize effectively to diverse transformation scenarios [50].

2.5 Longformer

Longformer [5], which stands for the Long-Document Transformer, is a modified version of the transformer [91] that is capable of processing lengthy documents. The success of transformers [91] in a variety of natural language tasks, including Transformer-XL [18] and GPT-2 [71] for generative language modeling and BERT [19] for discriminative language understanding, partly relies on the self-attention mechanism, which allows the network to comprehend contextual information from the entire sequence.

The transformer [91] model, while effective, has a significant limitation in that it can only process a limited number of tokens simultaneously. Thus, processing long documents poses a major challenge for the original transformer [91] model since it may require splitting the document into multiple segments of 512 tokens [41] or truncating the input [41]. In the former approach, the model processes each segment separately, and the predictions are then combined, but this method prevents the model from establishing connections between tokens in different segments due to the attention mechanism's inability to operate across segment boundaries. On the other hand, the latter approach suffers from information loss because of truncation. Therefore, Longformer [5] was developed with the goal of processing long documents in their entirety, rather than dividing them into segments. To achieve this, Longformer [5] introduces three different attention mechanisms, which are illustrated and compared to the original transformer [91] attention mechanism in Figure

2.10. Each of these mechanisms is explained in detail in the following paragraphs.

Longformer model is inspired by the other approaches that involve defining a sparse attention pattern and avoiding the computation of the full quadratic attention matrix multiplication. The attention pattern used in this model is most similar to that of the Sparse Transformer [13], which utilizes a dilated sliding window of blocks with a size of 8x8 provided by BlockSparse [27].

The classic transformer [91] model requires $O(n^2)$ memory when processing an input sequence of length n, as each token in the sequence can attend to all other tokens resulting in n^2 connections (it is important to mention that the transformer transformers the input sequence into a sequence with the same length). To address this issue, a sliding window attention mechanism is used, which is similar to convolutional neural networks (CNNs) [97]. With a sliding window of size w, each token can only attend to itself and its immediate neighbors within the window, reducing memory requirements to $n \times w$, or $O(n)$. However, this approach trades off some information as each token only attends to a limited range of information in a single layer of attention. To compensate for this loss, multiple stacked layers of sliding window attention are used to allow each token to gather information from a wider range. Nevertheless, this also involves a trade-off, as deeper layers may result in slower processing times [5]. Despite this, it is generally assumed that the most important information is located in the immediate neighborhood of each individual token [47].

The sliding window attention is merely one of the fundamental components of Longformer. The second essential component is the dilated sliding window (similar to dilated CNNs [90]). In the sliding window approach, a significant number of layers might be required to incorporate the entire sequence's attention, especially for longer sequences. To address this, the dilated sliding window component introduces a substantially larger attention window where the window has gaps size of dilation d, enabling faster integration of global information across layers [5].

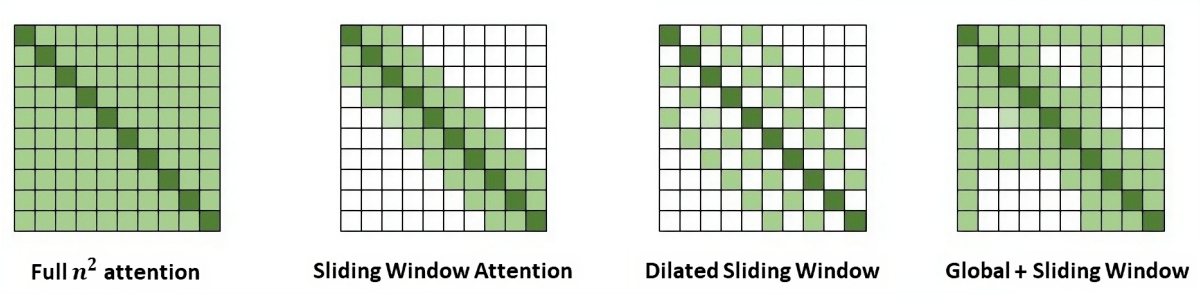


Figure 2.10: Different attention mechanisms introduced by Longformer compared to the original full attention introduced by transformers [91].

Given the layer structure in a transformer, the sliding window attention (fully local) can be utilized in the lower layers, while the dilated window can be employed in the higher layers. This approach is based on the assumption that local information is crucial in the lower layers to capture local features, while higher layers require more global information [5].

The third essential component of Longformer [5] is referred to as global+sliding window. In this approach, sparse global attention is employed where a few designated tokens from the sequence attend to all tokens at each layer, and any token in the sequence can attend to those designated tokens at each layer. This provides flexibility to select which tokens in the sequence are special. These special tokens can receive information from anywhere in the sequence at each layer and transmit information to any other unit in the sequence at each layer as well. The memory requirement for this component is $L \times (n \times w + s \times n \times 2)$, which is of $O(n)$, where L represents the number of transformer layers, n is the sequence length, w is the window size, and s is the number of special tokens. In this case n is multiplied by 2 to account for the fact that each special token can attend to any token in the sequence, and any token in the sequence can attend to the special tokens [5].

Longformer was pre-trained on the masked language modeling task, with pre-training continuing from the RoBERTa [52] released checkpoints, with minimal changes made

to adapt the longformer's attention mechanism. The window size used was 512 tokens, which is the size that a classic transformer-based model like RoBERTa [52] can handle as an entire document [5].

Chapter 3

Literature Review

Question-Answering (QA) is an interdisciplinary research area that merges various related fields, such as [2]:

- **Information Retrieval (IR)** is the process of retrieving relevant and useful information from a collection of data, such as text documents, images, or videos. Information retrieval systems can be used to search for specific information, such as a specific document or a particular piece of information within a document, or to provide a ranked list of results based on their relevance to a user's query. Examples of information retrieval systems include search engines, digital libraries, and enterprise search systems. The goal of information retrieval is to help users quickly and accurately find the information they need in a large and complex collection of data. However, these systems do not provide actual answers to questions, leaving it up to the user to extract the information they need from the returned documents or parts of a document. While the goal of QA systems is to provide specific answers to questions rather than just returning full documents or relevant passages, which is the current practice of information retrieval systems [2].

- **Information Extraction (IE)** is the process of automatically extracting structured information from unstructured data sources, such as text documents, web

pages, and databases. The goal of information extraction is to convert unstructured information into structured data that can be easily stored, analyzed, and used for various purposes, such as populating databases, generating reports, and supporting decision-making processes. For example, an information extraction system might extract named entities (e.g., people, organizations, locations) from a document and then organize this information into a structured format, such as a table or a database. The goal of information extraction is to save time and reduce the risk of errors associated with manual data entry, while providing structured information that can be easily processed and analyzed by computers. Information extraction systems often use techniques such as natural language processing (NLP), machine learning, and pattern recognition to identify and extract relevant information [81].

- **Natural Language Processing (NLP)** is a field of artificial intelligence and computer science concerned with the interaction between computers and humans using natural language. It deals with the ability of computers to understand, interpret, and generate human language. NLP techniques are used for tasks such as text classification, sentiment analysis, named entity recognition, machine translation, question-answering, and more. The goal of NLP is to enable computers to understand, interpret, and generate human language as naturally as possible, making it easier for people to communicate with machines and for machines to process and analyze vast amounts of human language data [81].

Question-answering focuses on creating systems that can automatically answer questions asked by people in a natural language. With various challenges related to syntax, semantics, and discourse, it has become a significant area of research in recent years. The field of question-answering research is diverse due to factors such as the range of information that can be used to answer a question, the type of question, answer type, source of evidence for answers, and answer retrieval modeling approach. Thus, in the

following sections, we explore the categorization of QA based on these parameters [63].

3.1 Categorization based on Scope or Domain of Accessible Information for Answering Question

Generally speaking, the task of question-answering is categorized into three different groups based on the answer and the scope of information it employs to find this answer. Focus of this thesis is on the extractive question-answering. However, the subsequent subsections provide individual explanations for each of these categories.

3.1.1 Extractive Question-Answering [93]

For the task of extractive question-answering, the inputs typically include:

1. A question, posed in natural language by a user.

2. A text corpus or document, which contains the information needed to answer the question. This can be a single document or a collection of documents, such as a database, a website, or a set of news articles.

3. Optional additional information, such as background knowledge or context, that can help the system determine the correct answer.

Based on these inputs, the extractive QA system will identify and extract the relevant sentences or phrases in the given text corpus that contain the answer to the question. The output will be the most relevant and accurate answer to the question, extracted from the text corpus. Extractive QA is often used in information retrieval and text summarization applications, where the goal is to provide quick and accurate answers to questions using existing text resources [93].

3.1.2 Open Generative Question-Answering [39]

The inputs for an open generative question-answering (OpenGQA) system typically include:

1. A question, posed in natural language by a user.

2. A text corpus or a knowledge base, which contains the information needed to answer the question. This can be a collection of documents, a database, a website, a set of news articles, or a structured knowledge graph.

3. Optional additional information, such as background knowledge or context, that can help the system determine the correct answer.

Based on these inputs, the OpenGQA system will generate an answer to the question by combining and synthesizing information from the text corpus or knowledge base, and any additional information available. The output will be a coherent and contextually appropriate answer to the question, generated by the system in natural language [39].

OpenGQA specifically refers to an open-domain QA system, meaning that it can answer a wide range of questions on any topic, rather than being limited to a specific domain or subject area. The system uses deep learning models, such as transformer-based neural networks, to generate the answers. The models are trained on large amounts of text data and are designed to understand the context, relationships, and patterns in the data, allowing them to generate coherent and contextually appropriate answers [39].

OpenGQA systems have the advantage of being able to answer questions that require interpretation, opinion, or judgment, as well as factoid questions. However, they can also be more challenging to develop and may not be as accurate as extractive QA systems for certain types of questions.

3.1.3 Closed Generative Question-Answering [39]

Closed Generative Question-Answering (ClosedGQA) is a type of question-answering system that generates answers to questions within a limited scope or domain. Unlike open-domain QA systems, which can answer questions on any topic, closed-domain QA systems are designed to answer questions within a specific subject area or domain, such as medicine, finance, or sports [39].

The inputs for a ClosedGQA system are similar to those of an open-domain QA system, including a question posed by a user, a text corpus or knowledge base, and any additional information or context that may be relevant. However, the system is trained on a smaller, more specific set of data that is relevant to the domain, allowing it to generate more accurate and relevant answers.

ClosedGQA systems are generally easier to develop and may perform better than open-domain QA systems for questions within the specific domain. However, they may not be able to answer questions outside of the domain or those that require knowledge from multiple domains.

3.2 Categorization based on Question Type

Additionally, the task of question-answering can be categorized into different classes based on the type of question. In our study, we opted for a straightforward approach of inputting a question and a passage to obtain the answer span from. However, in the following subsections we will describe other possible question-answering tasks, classified based on question type [63].

3.2.1 Multiple-Choice Question-Answering or MCQA

Answering questions with multiple options presents the challenge of equivocal reference [61], which, coupled with the difficulties of retrieving linguistic and relevant evidence,

can result in misleading solutions using standard information retrieval methods due to the presence of distracting, distorted options. Thus, the primary challenge in this setting is eliminating the misleading option [63].

Question 1: Can you tell me about the Mona Lisa painting?

Answer 1: The Mona Lisa is a 16^{th} century portrait painted by the Italian master Leonardo da Vinci. The subject of the painting is believed to be Lisa Gherardini, the wife of a wealthy Florentine merchant. The Mona Lisa is renowned for the enigmatic smile on the subject's face, which has captured the imagination of art lovers for centuries. The painting is housed in the Louvre Museum in Paris and is one of the most famous artworks in the world.

Question 2: Who is the subject of the painting?

Answer 2: The subject of the Mona Lisa painting is believed to be Lisa Gherardini, the wife of a wealthy Florentine merchant.

Question 3: Where is the painting housed?

Answer 3: The Mona Lisa painting is housed in the Louvre Museum in Paris.

Figure 3.1: An example of Conversational Question-Answering.

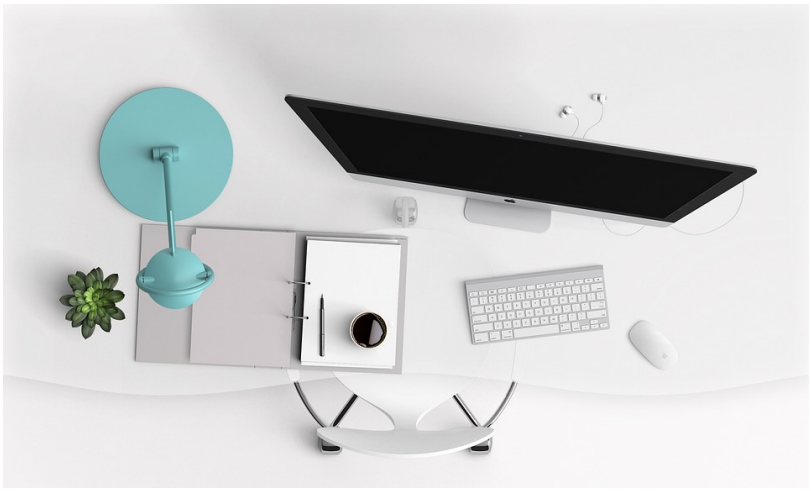
3.2.2 Conversational Question-Answering or CQA

In conversational question-answering the primary focus is on the conversation history. In other words, discourse is a crucial factor in answering conversational questions. Simply relying on context alone is insufficient as the answer to these type of questions also depends on the previous conversation [8, 68, 69]. The answer expected can be a simple fact (factoid question) or contain definitions and complex (non-factoid question). In

this case, the model should be able to understand the context of the conversation and provide relevant information in response to follow-up questions [63]. Figure 3.1 contains an example of CQA.

Question: How many dogs are there in the picture?

Answer: Two.



Question: What is on the desk to the right of the keyboard?

Answer: Mouse.

Figure 3.2: Visual Question-Answering Examples.

3.2.3 Visual Question-Answering or VQA

Visual question-answering involves answering questions based on visual content, which can be either an image or video. This task combines natural language processing and computer vision techniques [74,102] to generate answers that can be factoid, non-factoid, yes/no, or any other type. Recent advancements in the field of VQA [11] utilize a combination of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to convert the image and question features into a common representation [63]. Figure 3.2 shows two examples of VQA.

3.3 Categorizing based on Answer Type

Another method of categorizing current research in the field of question-answering is based on the type of answer that is expected, which is typically divided into two sub-categories: factoid and definition (or complex) question-answering. In this thesis we conducted most of our experiments on SQuAD [73], which is a dataset for factoid as well as non-factoid QA. Moreover, we briefly explored and conducted a limited number of experiments on NLQuAD [82], which is a dataset for non-factoid QA. Next subsections will explain the mentioned categories independently [63].

3.3.1 Factoid Questions Answering

Factoid questions are the questions that seek information about a straightforward fact or entity. For instance, *"What is the capital of India?"* can be answered with a single word, **"Dehli"**. Factoid questions typically require a brief and straightforward response [63].

3.3.2 Definition-Based or Non-Factoid Question-Answering

Definition-based question-answering [10, 49] involves finding a part of a given passage that answers the question at hand. For example, a question-answer pair such as *"Ques-*

tion: What is Machine Learning?" and *"Answer: Machine learning is a field that gives computer systems the ability to learn knowledge automatically"*, exemplifies non-factoid question-answering. The goal of this approach is to locate parts of a passage that fits the constraints of the question [63].

3.3.3 Hybrid Question-Answering

Answering hybrid questions, such as "What is the second largest planet in the solar system?" involves numerous challenges in natural language processing [100]. The response to such questions could be either factoid or non-factoid, but the primary challenge lies in selecting relevant information from a knowledge base or passage that satisfies the question's criteria. To achieve this, various semantic clues, such as the answer being a planet, being contained within the solar system, ranking second, and following a descending order, must be considered [63].

3.4 Categorization based on Evidence or Answer Source

Information can be stored either as unstructured raw text or as a knowledge graph. Depending on where the information is stored, we can categorize the type of question-answering into two subtypes: raw text-based QA and knowledge-based QA (KBQA). We are focusing on raw text-based QA datasets for this study. However, the following subsections represent an explanation for the mentioned classes [63].

3.4.1 Raw Text-Based Question-Answering

The difference in reasoning abilities between humans and machines becomes particularly obvious when answering questions from a passage [63]. Humans can effortlessly identify the relevant paragraph or sentences in a given passage, while this poses a challenge for machines [37, 44, 63].

3.4.2 Knowledge-Based Question-Answering

Information that is represented using an ontology is commonly referred to as a knowledge graph. In a knowledge graph, information can be depicted as a [subject, relation, object] triplet, where the relation shows the connection between the subject and object. The primary challenge here is to map the words in a question onto the graph and find the appropriate connections between nodes to obtain the answer [56]. Researchers have been drawn to this field due to the opportunity to progress from simple question-answer mapping [106] to breaking down the question into smaller parts and using deep learning models to process each one [36, 63].

3.5 Categorization based on Modeling Approach

The classification of QA models can be divided into rule-based, machine learning-based, and deep learning-based models. This ranges from early word matching models to the more recent transformer-based models. In this work, the main focus is on transformer-based models and enhancing their performance by augmenting additional information with them. Regardless, in the subsequent subsections, each of the noted categories will be defined in details [63].

3.5.1 Rule-Based Models

In most rule-based models, different rules are designed based on the type of WH question [76]. Language processing tasks such as semantic class tagging and entity recognition play a significant role in these systems. A rule can consist of a logical combination of any of these tasks. Each rule assigns points to all sentences of the input [43]. After all rules have been applied, the sentence with the highest score is returned as the answer. Currently, models for low-resource languages are mainly based on such rules [88], despite others have shifted to machine learning or deep learning-based systems, which have shown significant

performance improvement [63].

3.5.2 Machine Learning Based Models

In the majority of approaches utilizing machine learning models, the parsing result of a question is used for classification tasks like support vector machine (SVM), decision tree (DT), and naive bayes (NB). Other problems such as predicting whether a given community question will receive an answer or not, can also be tackled by machine learning through the use of predefined features sets [63].

3.5.3 Deep Learning Based Models

The advancement in QA research has shifted from traditional machine learning models to deep learning models, largely due to the increased availability of computing power and the introduction of Recurrent Neural Network (RNN) models in the text processing field. Currently, the leading technique for a question-answering task is to fine-tune pre-trained transformer models like Google's BERT [19] or OpenAI's GPT [9] [63].

3.6 A Few Examples of Previously Proposed Question-Answering Models

In the next subsections we will specifically describe two of the studied models during reviewing the literature, for the task of question-answering.

3.6.1 LSTM-based Deep Learning Models for Non-factoid Answer Selection [86] – An example of multiple choice question-answering

The authors of this article claim that previous investigations into the answer selection issue have utilized feature engineering methods, linguistic tools, external resources, and other approaches. Nevertheless, these approaches have several drawbacks. For instance, these models' reliance on additional resources may not be consistently possible, while models that employ linguistic tools are subject to systematic complexity. Instead, the authors propose a deep learning framework for the answer selection task that utilizes a bidirectional long short term memory, Bi-LSTM [26], model on both the question and the answer [86].

The answer selection problem is to find the best answer among the answer candidates $a_1, a_2, ..., a_k$ for a given question q. The correct answer to the given question might be only semantically related to the question and not lexically. This is the biggest challenge of this task. Furthermore, the answers provided can be noisy and may contain irrelevant information, further complicating the task [86].

There are three distinct categories of deep learning-based existing methods for the answer selection task, and the proposed framework in this study belongs to the first group [86]:

- First, the methods which learn the question and answer representation and measure the similarity between them using different similarity metrics [21, 24, 104].

- Second, methods in which a joint feature vector is constructed using both question and the answer and then the task of answer selection is converted to a classification or learning-to-rank problem [92].

- Third, the models that use textual generation for answer selection and generation

[4].

RNNs encounter issues with vanishing and exploding gradients, and they struggle with retaining long-term memory despite their effectiveness in capturing short-term memory. On the other hand, LSTMs [34] address these problems through the use of a context vector for long-term memory and a state (or hidden) vector for short-term memory, thereby reducing the gradient vanishing and exploding problem. However, single direction LSTMs [34] are limited in their ability to incorporate contextual information from future tokens [86].

A bidirectional LSTM [26] has the ability to maintain sequential context information in both directions by processing the sequence in two directions. With the Bi-LSTM [26] model, the input sequence is processed independently in both the forward and reverse directions, producing two separate sequences of LSTM output vectors. The overall output at each time step is computed using either max pooling, min pooling, or concatenation of the two output vectors from both directions. The framework then employs cosine similarity to measure the distances between the question and answer representations in order to identify the correct answer [86]. The fundamental architecture proposed by this paper is illustrated in Figure 3.3.

Furthermore, to enhance performance and preserve local linguistic information, the authors incorporate a Convolutional Neural Network (CNN) structure into the Bi-LSTM [26] architecture. Utilizing CNN filters on top of the Bi-LSTM [26] model results in improved embeddings for both the question and answer [86]. The architecture with this added CNN structure is illustrated in Figure 3.4.

Moreover, the authors proposed a straightforward attention model to differentiate the correct answer based on the given question. This approach has been employed previously in several other natural language processing tasks, including machine translation [4, 85], sentence summarization [77], and factoid question-answering [32, 83]. This model generates answer embeddings in accordance with the question context by utilizing both the

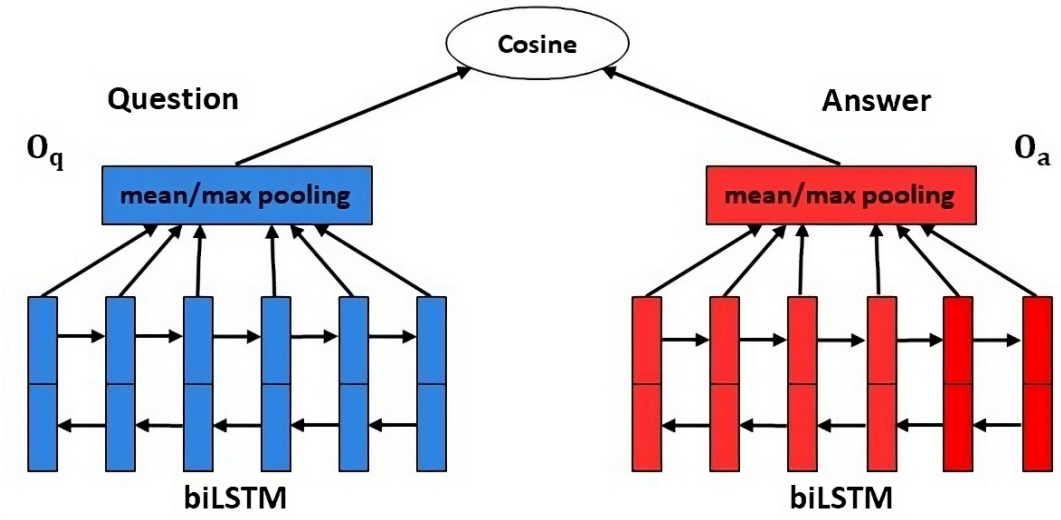


Figure 3.3: The basic Bi-LSTM [26] architecture proposed by this work [86] for answer selection task.

question representation and the answer choices provided. The attention mechanism assigns higher weights to words that are deemed more significant. In this mechanism, the Bi-LSTM [26] hidden vector of the answer is first multiplied by a coefficient, which is computed using the question's average pooling vector, and then updated to a new hidden vector. Finally, the original question representation and the updated answer's hidden vectors are inputted into the CNN or mean/max pooling layer [86]. Figure 3.5 illustrates this architecture.

The authors conducted an evaluation of their proposed approach on two datasets and discovered that concatenating the last vectors from both directions yielded the poorest performance. In addition, utilizing max pooling yielded better results than average pooling. Moreover, the introduction of the attention model enhanced the performance of this architecture. According to their final assertion, the experimental outcomes reveal that their proposed method outperforms numerous robust baselines [86].

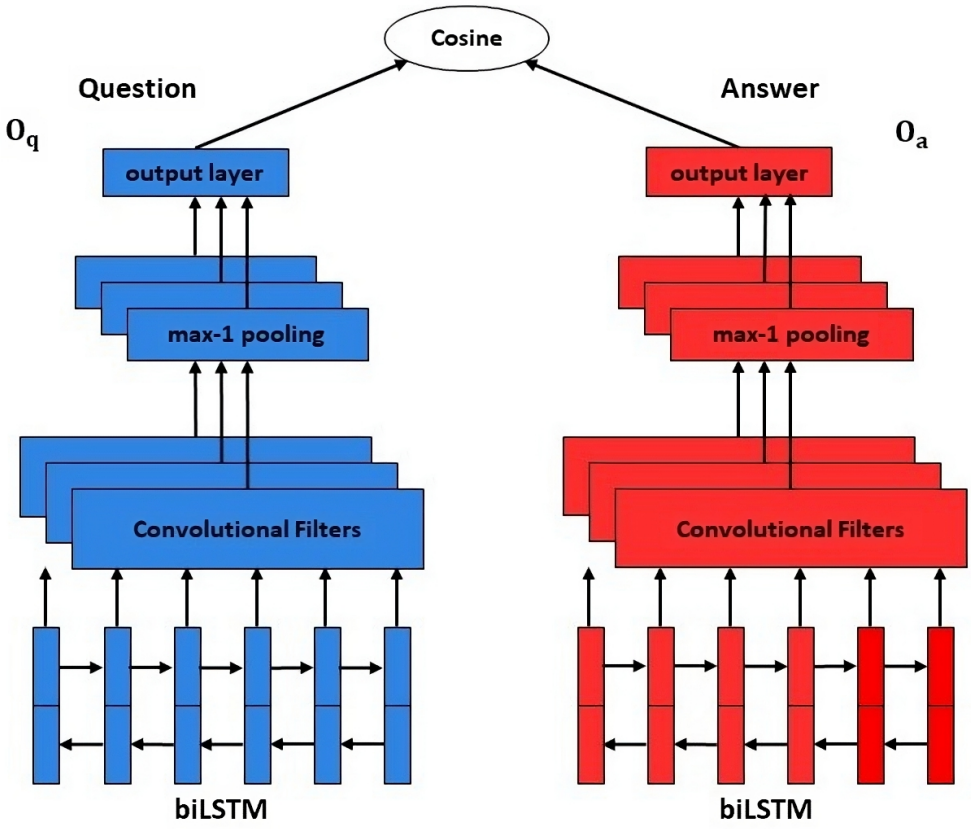


Figure 3.4: The Bi-LSTM/CNN architecture proposed by this work [86] for answer selection task.

3.6.2 Alignment over Heterogeneous Embeddings for Question-Answering [99] – An example of using information retrieval systems in question-answering

As stated by the authors of this paper, there are numerous neural deep learning [57] techniques available for the task of question-answering [79, 94, 95]. Therefore, other approaches, such as alignment methods [22, 84] that operate across various levels of representation, are no longer regarded as significant or effective. Furthermore, existing alignment methods do not consider contextualized word/sentence representations for the purpose of question-answering, causing them to underperform compared to supervised neural methods. For the first time, this paper proposes an unsupervised alignment ap-

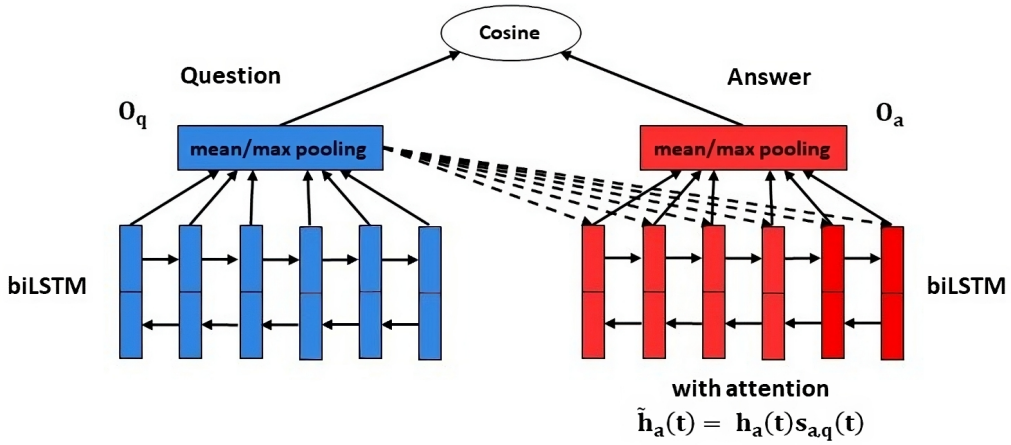


Figure 3.5: The Bi-LSTM [26] with attention architecture proposed by this work [86] for answer selection task.

proach for non-factoid question-answering that considers contextualized representations. This method models the underlying text at different levels of representation, including character, word, and sentence [99].

This paper presents two distinct formats for addressing the task of non-factoid question-answering [99]:

- **Multiple Choice Question-Answering** aims at identifying the correct answer to the given question from a set of possible answers using a supporting explanatory text that provides supplementary information obtained from an external knowledge base [99] (as depicted in Figure 3.6).

- **Answer Sentence Selection** aims at locating the sentence that includes the accurate information to answer the given question, among the available candidate answers that are structured as sentences [99].

As previously stated, various neural supervised techniques have been proposed for the question-answering task, utilizing stacked architecture and sometimes equipped with an attention mechanism, such as those discussed in [29, 79, 101]. While, some other approaches combine neural methods with query expansion techniques [58, 60, 62]. Nonethe-

Question: Which sequence of energy transformation occurs after a battery-operated flashlight is turned on?

1. electrical → light → chemical
2. electrical → chemical → light
3. chemical → light → electrical
4. **chemical → electrical → light**

Supporting paragraph(s): A chemical cell converts chemical energy into electrical energy; a flashlight chemical energy to light energy.

Figure 3.6: An example of a multiple-choice question along with the supporting paragraph extracted from an external knowledge base.

less, all these methods heavily rely on the selected training data and the creation of knowledge bases, which can be costly. Conversely, the method proposed by this paper is unsupervised and does not necessitate any training [99].

Alignment over Heterogeneous Embeddings (AHE) is the name given to the proposed approach, which leverages existing information retrieval (IR) components to identify the relevant paragraphs from the knowledge base (KB), based on the provided question and candidate answers. Subsequently, the algorithm aligns each word in the question and answer choices with the most pertinent word in the selected paragraphs to determine the correct answer. Consequently, the fundamental aspect of this methodology is the computation of the score for each candidate response via the alignment of two texts [99].

Regarding multiple choice question-answering, the first text consists of the question concatenated with each candidate answer individually, while the second text comprises the supporting paragraph, which consists of two or three sentences extracted from the

knowledge base. However, for the answer selection task, the first text is the given question, and the second text is each of the provided answer sentences [99].

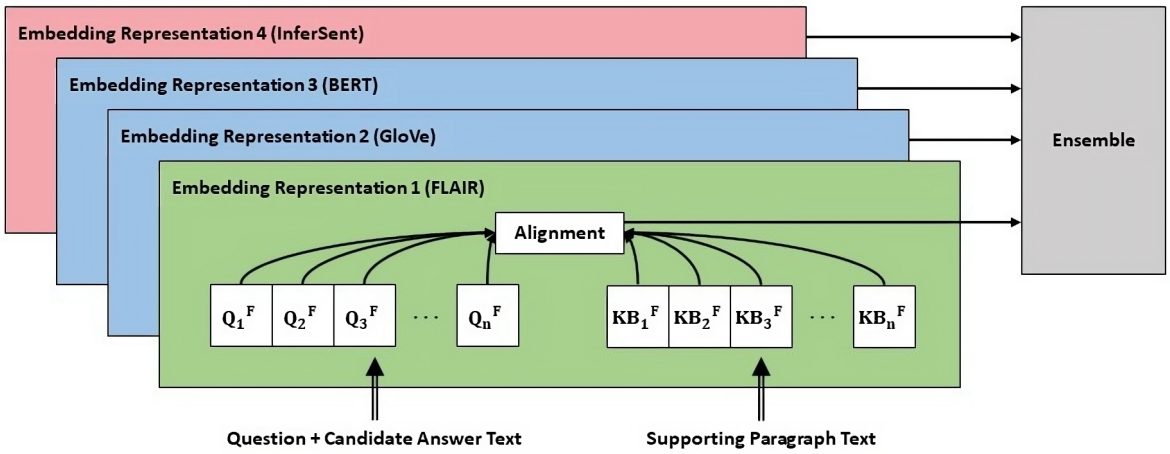


Figure 3.7: AHE [99] architecture for multiple-choice question-answering.

The alignment approach employed in both scenarios utilizes contextualized embeddings at three distinct levels of abstraction: character, word, and sentence. To generate the character-based embedding, the paper utilizes the FLAIR contextual character language model [1], which incorporates a Bi-LSTM [26] network to create character embeddings in both forward and backward directions, with the outputs from both directions concatenated at the end. Moreover, two different word-based embedding techniques are utilized: bidirectional encoder representations from transformers (BERT [19]) and GloVe [66]. Sentence-based representation is created using InferSent [16] embeddings. Finally, the unsupervised variant of the NoisyOr formula is employed to aggregate the scores of candidate answers over these four distinct embedding representations [99].

$$NosiyOr_M(i) = 1 - (\prod_{m=0}^{M}(1 - \alpha^m \times S_i^m)) \tag{3.1}$$

This calculation determines the total score for answer candidate i. In this equation, M represents the total number of representations, which is four in this particular scenario. S_i^m denotes the score of answer candidate i in representation m. Lastly, α^m is a

hyperparameter used to adjust high distributions of answer probabilities [99].

Additionally, the paper implemented a meta-classifier to directly learn the aggregation function from the data. This meta-classifier contains two fully connected dense layers with hidden sizes of 16 and K, respectively, where K is the maximum number of candidate answers for a given dataset. The activation function used in the first dense layer is tanh, while softmax is applied to the second output layer [99].

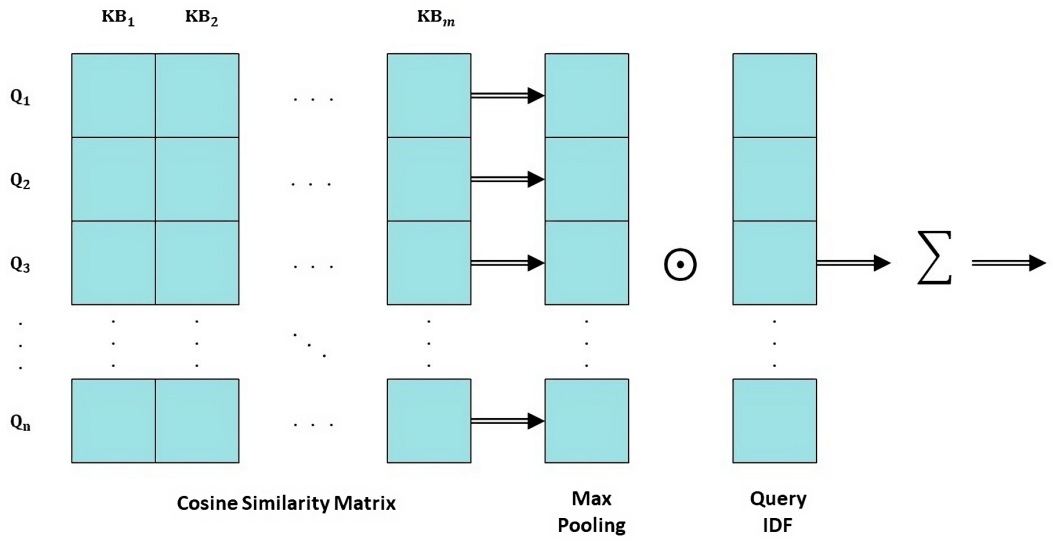


Figure 3.8: Alignment score computation of AHE [99].

To compute alignment scores for character-based and word-based representations, cosine similarity is used on the embedding vectors of the input query tokens Q_i and the supporting knowledge base paragraph tokens KB_j. The resulting similarity matrix is then fed through a max-pooling layer to find the most relevant supporting paragraph for each query token. The resulting max-pooled vector is multiplied by the IDF (inverse document frequency) of the query tokens, and the results are summed to produce the overall alignment score for a given query Q_a (question Q concatenated with candidate answer a) and supporting paragraph P_j. Figure 3.8 illustrates the alignment component of AHE [99] architecture for character and word representations. For sentence-based representation, the alignment score between the query Q_a and the supporting paragraph

P_j is computed as the dot product between the two, which is then normalized using softmax across all candidate answers [99].

AHE [99] was evaluated on the two different QA settings mentioned earlier, and the results indicate that it has a consistent and robust performance across all three studied datasets. Additionally, the authors reported that AHE outperforms many neural architectures, including some of the RNN-based models that incorporate augmented attention mechanisms. Ultimately, they concluded that using a combination of two or more embedding representations is more effective than relying on a single embedding representation alone [99].

Chapter 4

Methodology

This study focuses on transformer-based models for extractive question-answering, with the objective of evaluating their performance and suggesting a feature-augmented architecture to enhance their effectiveness.

4.1 Transformer-Based Models for the task of Extractive Question-Answering

As previously stated, all transformer-based language models undergo pre-training on unsupervised tasks such as masked language modeling and next sentence prediction, before being fine-tuned on various downstream tasks using task-specific data. When examining these architectures for the purpose of extractive question-answering, it can be observed that there is an additional linear layer placed on top of the base models to identify the span of the answer. The input to each of theses transformer-based models is a single packet consisting of the question and the passage, containing the answer to the given question, separated by a [SEP] token, and the outputs are the start and end positions of the answer span [19].

For the task of question-answering, in the fine-tuning step, the mentioned models use

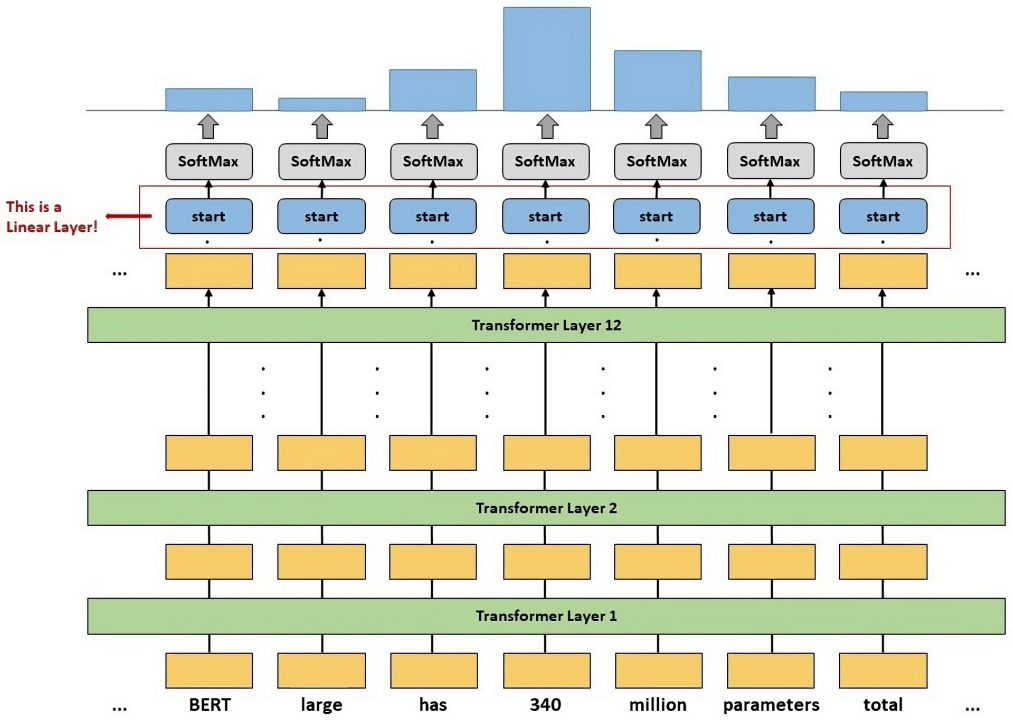


Figure 4.1: Start of answer index prediction in BERT model. Start is a vector of weights, which is learned by the last linear layer used on top of BERT to classify the start and end of answer. The same weights are applied to every position. Token with highest probability is chosen as the start of answer.

Start Token Classifier Vector (S) and *End Token Classifier Vector* (E) to predict the beginning and end of the answer span, respectively. These two vectors are only generated during the fine-tuning stage. The model then calculates the dot product between the last hidden states (final embeddings) and S vector, and applies a softmax activation function to the results of these dot products, to determine the highest probability for the start of the answer. Similarly, to predict the end of the answer span, the model follows the same process using the E vector [19].

The probability of word i being the start of answer span is calculated using this formula [19]:

$$P_i = \frac{e^{S.T_i}}{\sum_j e^{S.T_j}} \tag{4.1}$$

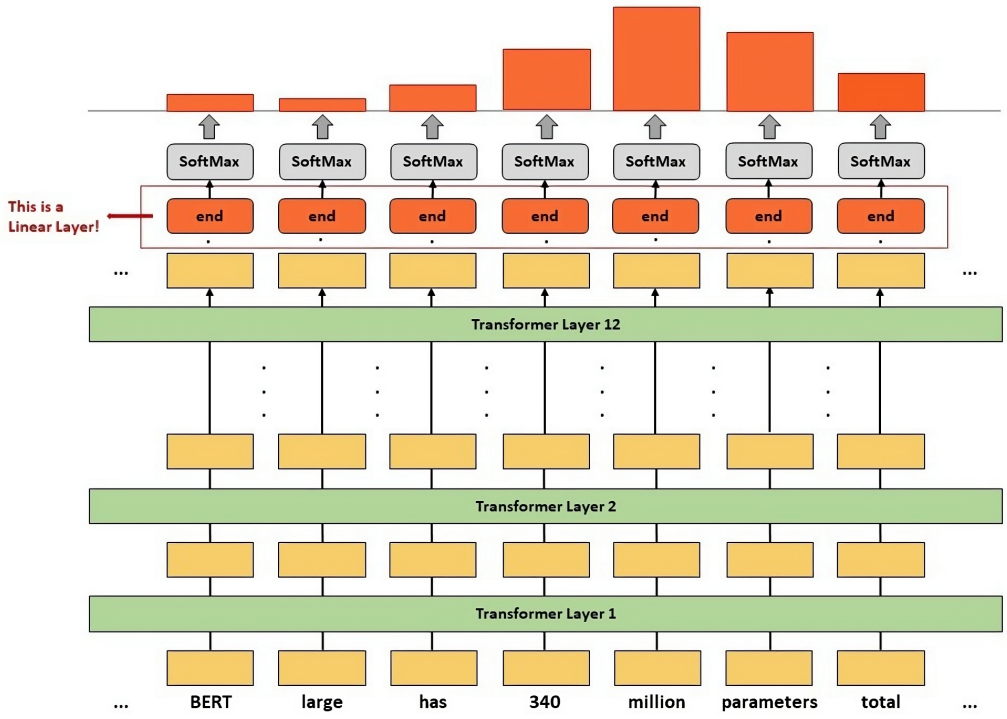


Figure 4.2: End of answer index prediction in BERT model. Same as the start vector, end is a vector of weights, which is learned by the last linear layer used on top of BERT. The same weights are applied to every position. Token with highest probability is picked as the end of answer.

In this formula T_i is the final embedding of the token i and . specifies dot product.

While, the probability of word k being the end of answer span is [19]:

$$P_k = \frac{e^{E.T_k}}{\sum_j e^{E.T_j}} \tag{4.2}$$

Finally, the score of a candidate span starting at position i and ending at position k, where $k \geq i$, is $S.T_i + E.T_k$. The goal is to predict the answer span with the highest score as the correct answer to the given question [19].

The pre-trained versions of all the models for the task of questions answering are provided by the transformers package on Huggingface website. These pre-trained models come equipped with a span classification head on top of the base model, enabling them to perform extractive question-answering tasks like question-answering using SQuAD [73].

Thus, as mentioned previously, the span classification head consists of a linear layer on top of the hidden-states output, which generates span start and span end logits. However, in order to enhance the base model in all of the studied transformer-based models, we modified the architecture in order to incorporate additional linguistic features into each model.

4.2 Feature-Augmented Architecture

Despite the performance metric of transformer-based models being high (in most of the downstream natural language tasks), basic Natural Language Understanding (NLU) errors are still present, particularly when the linguistic structures are complicated and the model faces complex logic. NLU error is a type of error that occurs in the processing of natural language input. It refers to a failure in correctly interpreting the meaning or intent behind a spoken or written sentence. This can happen when the language model used for NLU does not have enough context or data to accurately understand the input, or when the input is poorly formed or unclear. As a result, the output generated by the NLU system may be incorrect or not aligned with the intended meaning, leading to an NLU error [105].

This motivates us to create a model that integrates additional linguistic features to enhance the ability of the studied transformer-based models to comprehend linguistic structures, reducing the number of NLU errors compared to the predictions of the base models.

The architecture we propose involves augmenting linguistic features and draws inspiration from the work on feature engineering by [105]. However, our contribution is the extension of this framework to various transformer-based models, as well as a few modifications to the original architecture presented by [105] resulting in different variants of this architecture. The backbone model for our proposed architecture can be any of

the studied transformer-based natural language models (i.e. BERT [19], RoBERTa [52], BART [50] and Longformer [5]). We utilized the *SpaCy* NLP package to extract linguistic features from the context passage and question.

4.2.1 Linguistic Features

The utilized subset of linguistic features include token-level features for both questions and answers. We selected token-level features to align with the token-level input used in transformer-based models architecture. The features set for each token includes four attributes. In this work we use off-the-shelf feature extraction tools, however, it is possible to extend this features set.

Named Entity Recognition

Named Entity Recognition (NER) is a subtask of Natural Language Processing (NLP) that involves identifying named entities in a text, such as person names, companies, organizations, locations, products and so on. NER is an important step in the process of text analysis, as it allows for the extraction of structured information from unstructured text data. For example, a NER system can be used to extract and categorize names, places, and organizations from news articles or social media posts. The results of NER can then be used for various purposes, such as information retrieval, relationship extraction, and text classification. NER systems can use various techniques, including rule-based methods, machine learning and deep learning methods, to perform the recognition task.

Part-of-Speech Tag

Part-of-Speech (POS) tagging is a task in Natural Language Processing (NLP) that involves marking each word in a text with its corresponding word class or part-of-speech. A word class is a category that a word belongs to based on its grammatical properties and its role in a sentence.

There are several common categories of parts-of-speech, including *nouns*, *verbs*, *adjectives*, *adverbs*, *prepositions*, and *pronouns*. In English, for example, nouns are words that refer to objects, people, places, and ideas, verbs are words that express actions or states, and adjectives are words that describe nouns.

POS tagging is a crucial task in NLP as it helps to disambiguate words and provides information about the grammatical structure of sentences. This information is often used as an input for other NLP tasks, such as Named Entity Recognition (NER), Parsing, and Text Classification, among others.

Syntactic Dependency

Syntactic dependency (DEP) refers to the relationship between the words (or tokens) in a sentence and how they are connected to each other to form a complete and grammatically correct sentence. In syntactic dependency, words in a sentence are represented as nodes and the relationships between the words are represented as directed edges or arcs. The directed arcs capture the syntactic relationships between words, such as subject-verb, object-verb, adjective-noun, etc.

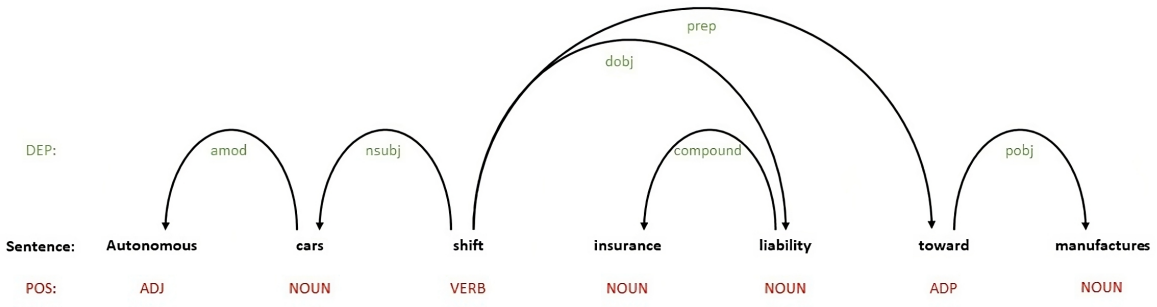


Figure 4.3: An example of dependency tree, showcasing the syntactic interrelations among words within a sentence. The red labels signify the POS attributes of individual words while the green labels correspond to the DEP features.

The illustration presented in Figure 4.3 exemplifies the syntactic dependencies among the words in a sentence. In this example, the relationship between the words *cars* and *autonomous* is identified as an adjectival modifier, abbreviated as **amod**, which denotes an adjectival phrase that serves to modify a noun (or pronoun). Additionally, the relationship between the words *shift* and *cars* is categorized as a nominal subject, or **nsubj** for short, which represents a nominal element that serves as the syntactic subject and the proto-agent of a clause. Moreover, the relationship between the words *shift* and *liability* is labeled as a direct object, also known as **dobj**, which refers to the noun phrase that functions as the accusative object of the verb. Ultimately, the DEP feature is assigned to the head node by associating its corresponding edge label. Some words might not be mapped to any DEP features when there is no syntactic relationship pointing towards them. For instance, DEP feature for the word *Autonomous* is **amod** and there is no DEP feature mapped to the word *shift*.

Syntactic dependencies provide valuable information about the grammatical structure of a sentence and can be used to understand the relationships between words, to identify the main subjects and objects in a sentence, and to determine the role of each word in a sentence. This information can be useful in various NLP applications, such as Parsing, Text Classification, Question-Answering, and many others.

Stop Words

Stop words (STOP) are common words in a language that are often filtered out before natural language processing because they carry little to no meaning and are considered to be of little value for most NLP tasks. Examples of stop words in English include *a*, *an*, *the*, *and*, *in*, *of*, etc. The choice of stop words can depend on the specific NLP task, the language being processed, and the domain or subject matter of the text being analyzed.

Stop words are often removed from a text corpus to reduce its size, speed up processing times, and improve the results of NLP tasks by eliminating irrelevant information. The

list of stop words can vary, but typically includes high-frequency function words such as articles, prepositions, conjunctions, and pronouns.

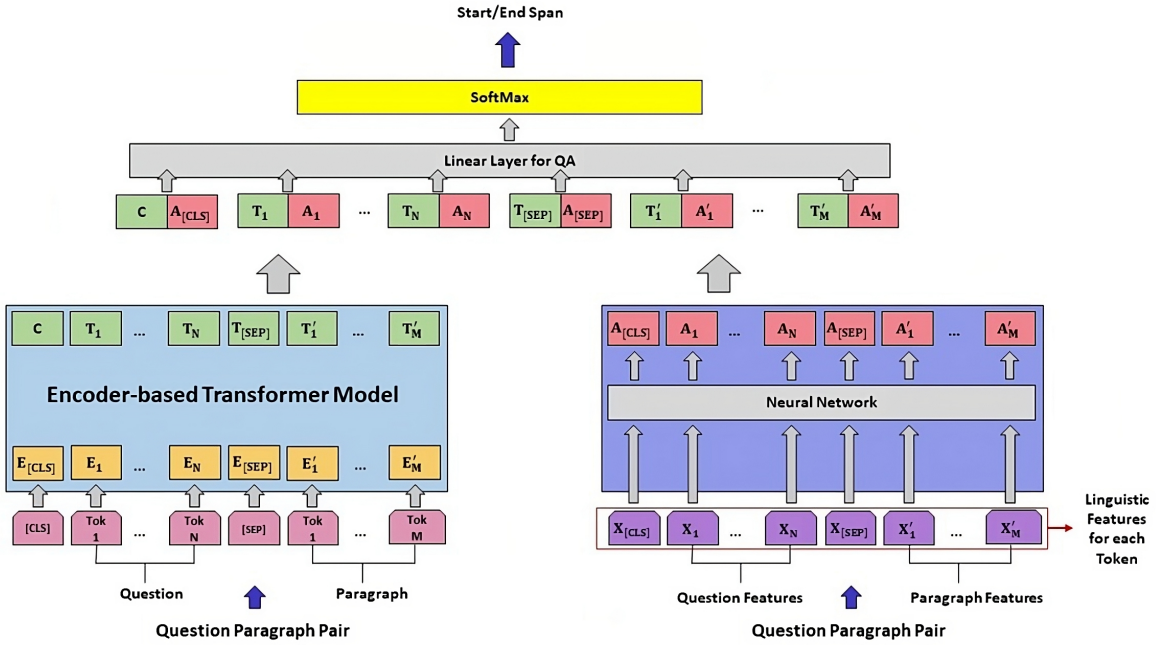


Figure 4.4: *General Feature Augmentation Architecture.* The component on the right side first extracts the linguistic features from question-passage pair then modifies them using a neural network.

4.2.2 Variants of Feature-Augmnted Architecture

General Feature Augmentation Architecture

The outputs of named entity recognition (NER), part-of-speech (POS), and dependency parsing (DEP) are presented in the form of string labels. To employ these features as inputs, we perform a conversion of these string labels into integer values. For instance, we convert 18 distinct name entity labels into integers ranging from 1 to 18. The last feature, STOP, is a true/false binary value that we map to 1 or 2, respectively, without any additional processing. Given a paragraph context and a question, we extract these four features from both context passage and question, which are tokenized using BERT's

[19] tokenizer. For special tokens, such as [CLS] and [SEP], we consider the value of 0 for each of these four features. The resulting vector is then padded to match the maximum sequence length (max_seq_len), which is the same size as the backbone model's maximum sequence length. These features are subsequently passed into a specific neural network, which produces a vector that encompasses aggregated information from all of these features.

The outputs generated by the transformer-based model and the linguistic features component are merged to create an input for the last linear layer, located on top of the transformer-based models for answer span detection. The output of this linear layer will be passed through a softmax layer. The outcome of the softmax serves as the start and end of the answer span. The proposed general architecture for integrating features with transformer-based models is depicted in Figure 4.4.

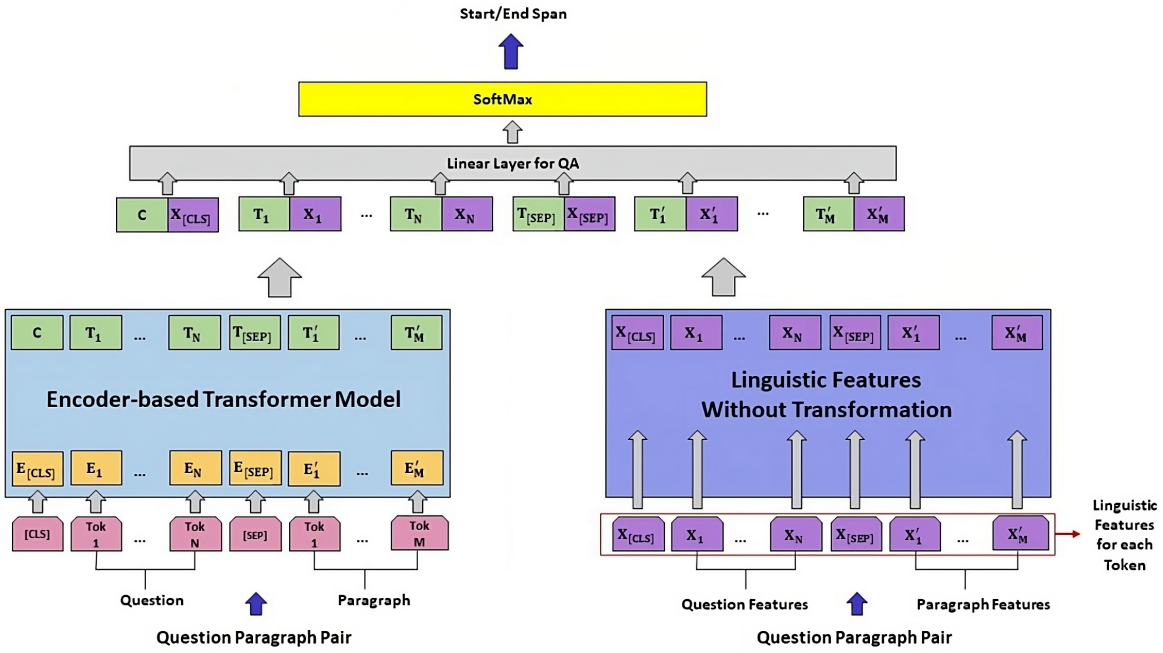


Figure 4.5: *Direct Feature Augmentation Architecture.* In this variant of the architecture the neural network component is removed from the *General Feature Augmentation Architecture.*

Direct Feature Augmentation Architecture

A feasible alternative of the general architecture is to utilize the extracted features directly, without feeding them into any neural network and remove the neural network component completely from the general architecture (shown in Figure 4.5). That is, we concatenate the extracted feature vectors with the last hidden layer vectors calculated by the transformer-based model.

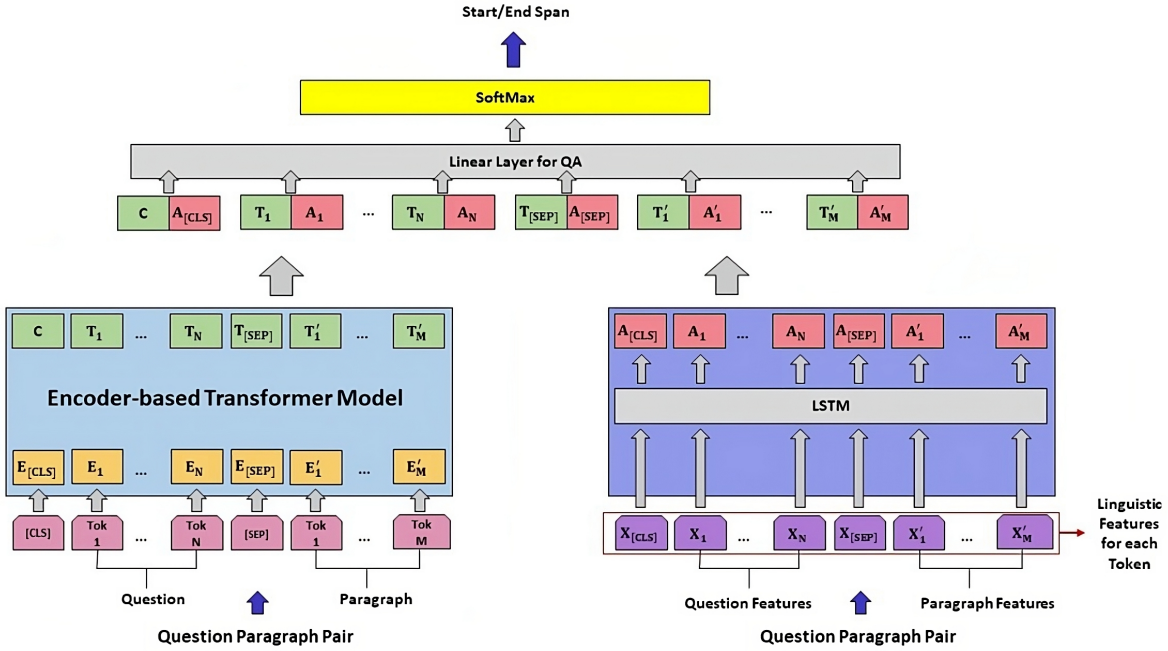


Figure 4.6: *LSTM Feature Transformation Architecture.* In this variant of the architecture the neural network component from the *General Feature Augmentation Architecture* is replaced with an LSTM [34] layer to transform the features.

LSTM Feature Transformation Architecture

Moreover, the neural network component from the general architecture can be replaced with a Long-Short Term Memory (LSTM, [34]). Figure 4.6 illustrates this variation of the base architecture. That is, we concatenate the modified feature vectors (through an LSTM [34] layer) with the last hidden layer vectors produced by the transformer-based

model.

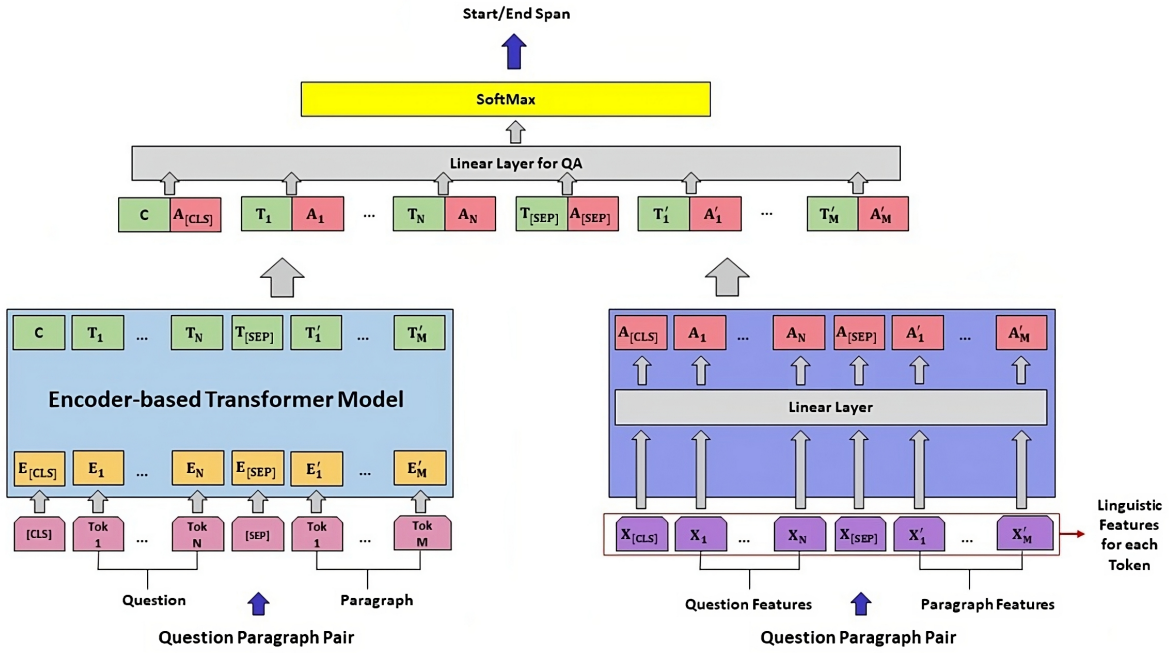


Figure 4.7: *Linear Feature Transformation Architecture.* In this variant of the architecture the neural network component from the *General Feature Augmentation Architecture* is replaced with a linear layer to transform the features.

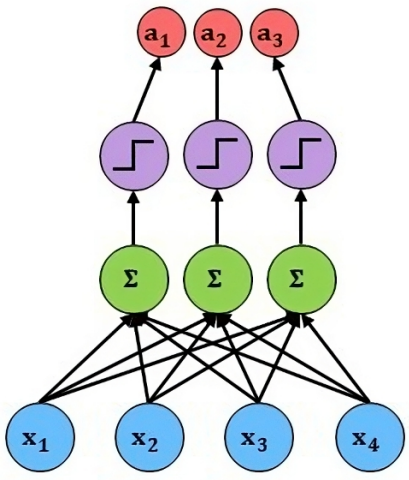


Figure 4.8: Linear Layer used in the *Linear Feature Transformation Architecture.*

Linear Feature Transformation Architecture

Additionally, the neural network element within the linguistic feature component in the general architecture can be substituted by a basic linear layer succeeded by a rectified linear unit (ReLU) activation layer, as illustrated in Figure 4.7. This thesis employed a 4-by-3 linear layer architecture, depicted in Figure 4.8, for aggregating information from all the features.

Chapter 5

Experiments and Results

In this Chapter, we report, compare and discuss the experimental results of different transformer-based models as well as the results of the proposed feature-augmented transformer-based architecture and its different variants. First, we will give a brief summary for the two question-answering datasets that we used. Furthermore, we will cover and explain all the performance metrics that are commonly used to evaluate question-answering models. Moreover, we will discuss and compare the performance of proposed architecture for different question types. Additionally, an ablation study will be performed on one of the studied question-answering datasets to investigate the impact of particular features on the proposed feature-augmented transformer-based architecture. Finally, some case studies will be provided demonstrating the efficacy of some particular employed linguistic features.

5.1 Stanford Question Answering Dataset [73]

SQuAD [73], which stands for Stanford Question Answering Dataset, is a popular benchmark dataset for evaluating question-answering models. It consists of a large set of questions (over 100,000) that are based on a diverse set of texts, along with their corresponding ground truth answers [73].

The text passages in this dataset come from a diverse set of sources, including news articles, Wikipedia pages. For each passage, crowdworkers were assigned the duty of generating up to 5 questions and answering them using the content of passage. These questions were required to be entered into a designated text field, while the corresponding ground-truth answers had to be highlighted within the passage. Furthermore, the workers were prompted to ask questions utilizing their own phrasing, without using the words from the paragraph [73].

Previously available datasets for reading comprehension suffer from one of these two issues:

- Datasets that are of high quality, such as MCTest [75] and ProcessBank [6], are too small to train modern data-intensive models. Data-intensive refers to a type of computing or data processing that involves handling and analyzing large amounts of data, often in parallel or distributed computing environments.

- Large datasets, such as those proposed and used by [32] and [33], are semi-synthetic and do not have the same features as explicit reading comprehension questions.

SQuAD dataset was introduced as a solution to the demand for a comprehensive reading comprehension dataset that is both large in size and high in quality [73].

This dataset is unique in that it requires models to not only generate an answer to a question but also to locate the answer within the given text. This is because the questions are designed to be answerable by selecting a span of text from the corresponding passage. Therefore, SQuAD [73] is often used to evaluate models that perform both reading comprehension and text understanding tasks [73].

SQuAD [73] has been used to evaluate a variety of models, from traditional rule-based systems to modern deep learning models. The current state-of-the-art models for SQuAD [73] use deep learning techniques such as transformers [91] and other transformer-based models (e.g. BERT [19]).

Passage: The Normans (Norman: Nourmands; French: Normands; Latin: Normanni) were the people who in the 10th and 11th centuries gave their name to Normandy, a region in **France**. They were descended from Norse ("Norman" comes from "Norseman") raiders and pirates from **Denmark, Iceland and Norway** who, under their leader Rollo, agreed to swear fealty to King Charles III of West Francia. Through generations of assimilation and mixing with the native Frankish and Roman-Gaulish populations, their descendants would gradually merge with the Carolingian-based cultures of West Francia. The distinct cultural and ethnic identity of the Normans emerged initially in the first half of the **10th century**, and it continued to evolve over the succeeding centuries.

Question 1: In what country is Normandy located?

France

Question 2: From which countries did the Norse originate?

Denmark, Iceland and Norway

Question 3: What century did the Normans first gain their separate identity?

10th century

Question 4: What is France a region of?

Not answerable

Figure 5.1: An example of question-answer pairs for a sample passage from the SQuAD [73] dataset. Each of the answers is a span of text extracted from the passage.

SQuAD [73] has also been used as a benchmark for transfer learning, which is a technique for training models on one task and then applying the learned knowledge

to another related task. Transfer learning has been shown to be highly effective for improving the performance of question-answering systems on SQuAD [73] and other similar natural language processing tasks. All these models have helped to advance the state-of-the-art in NLP and have led to many new applications of question-answering technology in fields like customer service, education, and healthcare [73].

Furthermore, there are two versions available for this dataset:

- **SQuAD 1.1** is the initial release of SQuAD [73] and it encompasses more than 100,000 question-answer pairs derived from over 500 articles.

- **SQuAD 2.0** merges the question-answer sets from its predecessor version with more than 50,000 unanswerable questions that resemble answerable ones, created through adversarial writing by crowdworkers. Consequently, any system utilizing this dataset must be capable of not only predicting answers when feasible, but also recognizing instances where the paragraph does not provide support for an answer, and refraining from answering in such cases. Figure 5.1 contains an example of question-answer pairs for a passage taken from SQuAD 2.0.

As apart of the present study, we aim to investigate the efficacy of individual transformer-based models and the proposed feature-augmented architecture across different question types (e.g. What, Where, When, etc.) that are present in SQuAD [73] dataset. Accordingly, we classified the dataset based on the type of question in order to illustrate their respective distribution. The distribution percentages of each question type in SQuAD [73] dataset are depicted in Figure 5.2.

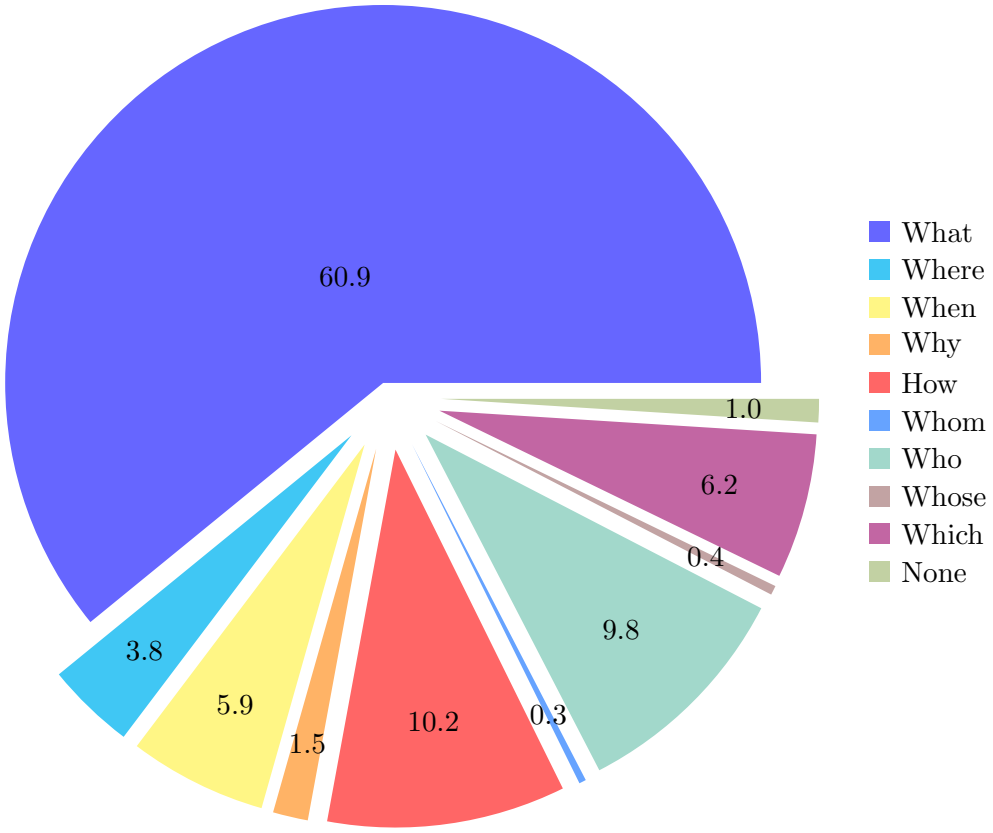


Figure 5.2: Different question types in SQuAD [73] dataset.

5.2 Non-Factoid Long Question Answering Dataset [82]

NLQuAD [82], which stands for Non-Factoid Question Answering Dataset, is the first dataset that establishes baseline approaches for long-passage non-factoid question-answering, a challenging task that demands a comprehensive understanding of language at the document level. Unlike other question-answering datasets that focus on span detection, NLQuAD [82] features non-factoid questions that cannot be answered by a brief text segment and require responses which encompass multiple sentences, including descriptive answers and opinions [82].

This dataset contains 31,000 non-factoid questions and their corresponding long answer passages, sourced from 13,000 news articles from the BBC. The questions and answer

passages are extracted from the sub-headings and subsequent body paragraphs of these articles, respectively [82].

Question: How are people coping in the lockdown?

Passage: China has widened its travel restrictions in Hubei province - the centre of the coronavirus outbreak - as the death toll climbed to 26. The restrictions will affect at least 20 million people across 10 cities, including the capital, Wuhan, where the virus emerged. On Thursday, a coronavirus patient died in northern Hebei province - making it the first death outside Hubei. [...] We now know this is not a virus that will burn out on its own and disappear. [...] And we still don't know when people are contagious. Is it before symptoms appear, or only after severe symptoms emerge? One is significantly harder to stop spreading than the other. [...] **One doctor, who requested anonymity, describes the conditions at a hospital in Wuhan. [...] "I was planning to stay in my apartment because I'm scared to go to the gym, and I'm scared to go to out in public, and not many people are willing to go out." (141 words).** Vietnam and Singapore were on Thursday added to the nations recording confirmed cases, joining Thailand, the US, Taiwan and South Korea. [...] Taiwan has banned people arriving from Wuhan and the US state department warned American travellers to exercise increased caution in China. (passage length: 921 words)

Figure 5.3: An example of question-answer pairs from NLQuAD [82]. The correct answer span is bolded within the passage.

In the majority of existing question-answering datasets, such as SQuAD [73], questions are generated by crowdworkers based on provided short passages, and answers are extracted from these passages. This process of question generation can result in question-answer pairs that are too easy, as models can simply rely on shallow pattern

matching to detect the most relevant text span for a given question [46]. In contrast, NLQuAD [82] annotations are automatically generated directly from the news articles themselves, without any human input [82].

Furthermore, NLQuAD [82] stands out from other long-context question-answering datasets, such as MS MARCO [59] and ELI5 [23], in that it does not use information retrieval methods to identify supporting documents. The retrieved documents in these datasets may not always contain all the facts needed to answer a question or may only be related to the question but not provide a direct answer [82].

As mentioned previously, NLQuAD [82] necessitates a comprehensive understanding of language at the document level, which is challenging due to the average length of the documents and answers being 877 and 175 words, respectively. These exceed the maximum input length of some current state-of-the-art QA transformer-based models, like BERT [19] and RoBERTa [52], due to their computational and memory requirements. Therefore, it is impractical to train and assess (document, question, answer) tuples utilizing these models in an end-to-end fashion and some adjustments are required to make these model compatible with long question-answering [82].

Furthermore, the questions provided by NLQuAD are typically not self-contained. For instance, to answer the question *"How are people coping in the lockdown?"* (Figure 5.3), the question-answering model must read the document to comprehend the idea of **lockdown** first and then locate information on people's behavior in lockdown [82].

In this thesis, we conducted most of our experiments on SQuAD [73] since this is a well-known benchmark dataset for question-answering task. Moreover, we conducted some experiments on NLQuAD [82] to investigate the effect of the proposed feature-augmented architecture on a non-factoid, long question-answering dataset as well.

5.3 Performance Metrics for Question Answering

5.3.1 Precision

In natural language processing (NLP), precision is a measure of the accuracy of a system or model's predictions. It measures the proportion of true positives (correctly identified instances) out of all instances that were predicted as positive (i.e. both true positives and false positives).

The formula for calculating precision is:

$$Precision = \frac{TP}{TP + FP} \tag{5.1}$$

Here's a breakdown of the terms used in the formula:

- **True positives (TP):** Instances that were correctly identified by the model as positive.
- **False positives (FP):** Instances that were incorrectly identified by the model as positive.

In question-answering models, which extract the span of answer, precision is still a measure of the accuracy of the model's predictions. However, it is usually calculated slightly different from other NLP tasks.

For calculating precision for each instance in our dataset, we first standardize the format of both the predicted answer and the ground truth. This involves removing articles and punctuation, standardizing white spaces (removing extra white spaces), and converting the text to lower case. This process is known as normalization. Then we use the following formula to compute precision for one single sample of our dataset:

$$Precision = 1.0 \times \frac{num\_same}{len(pred\_tokens)} \tag{5.2}$$

Here's a breakdown of the terms used in the formula:

- **num_same:** Number of common tokens between the normalized ground truth and normalized predicted answer.

- **len(pred_tokens):** Length of normalized predicted answer, or in other words the number of tokens in the normalized predicted answer.

Finally, we report the average of precision scores calculated for all of the instances in our evaluation set as the precision score of the model.

5.3.2 Recall

In natural language processing (NLP), recall is another commonly used evaluation metric that measures the proportion of true positives (correctly identified instances) out of all actual positive instances in the dataset (i.e. both true positives and false negatives).

The formula for calculating recall is:

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

Here's a breakdown of the terms used in the formula:

- **True positives (TP):** Instances that were correctly identified by the model as positive.

- **False negatives (FN):** Instances that were missed by the model and were incorrectly identified as negative.

Same as precision, in question-answering models, recall is usually calculated slightly differently than in other NLP tasks. However, it is still a measure of the model's ability to correctly answer a question given a context

For calculating recall for each instance in our dataset, we first normalize both the predicted answer and the ground truth (we explained the normalization process previously in subsection 5.3.1). Then we use the following formula to compute recall for one

single sample of our dataset:

$$Recall = 1.0 \times \frac{num\_same}{len(truth\_tokens)} \tag{5.4}$$

Here's a breakdown of the terms used in the formula:

- **num_same:** Number of common tokens between the normalized ground truth and normalized predicted answer.
- **len(truth_tokens):** Length of the normalized ground truth, or in other words the number of tokens in the normalized ground truth.

Finally, we report the average of recall scores calculated for all of the instances in our test set as the overall recall score of the model.

5.3.3 F1 Score

In natural language processing (NLP), F1 score is a commonly used evaluation metric that provides a balanced measure of a system's precision and recall. F1 score is the harmonic mean of precision and recall, and it is often used as an overall evaluation metric for NLP tasks.

The formula for calculating F1 score is:

$$F1\_Score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{5.5}$$

In question-answering, F1 score is also a commonly used evaluation metric, and it is calculated using the similar approach and formula as in other NLP tasks.

We calculate the F1 score of each instance individually and then take the average of all F1 scores to derive the overall F1 score of the model.

Furthermore, we consider the scenario where neither the predicted answer nor the ground truth contains an answer (i.e. $len(pred\_tokens) = 0$ and $len(truth\_tokens) = 0$) as having an F1 score of 1. However, if either one of them is empty, we assign an F1

score of 0. Additionally, if the predicted answer generated by the model and the ground truth have no common tokens, according to the given formulas, the precision and recall will both be 0, resulting in an F1 score of 0 as well.

5.3.4 Exact Match

Exact match is a common evaluation metric used to measure the performance of a model in the question-answering task. Exact match (EM) measures the percentage of questions for which the model provides an exact and complete answer that completely matches the ground truth answer.

For calculating exact match for each instance in our dataset, we first normalize both the predicted answer and the ground truth (we explained the normalization process previously in subsection 5.3.1). Then, based on the following conditions we determine the exact match for each sample in our dataset:

- If the normalized predicted answer matches the normalized ground truth answer exactly (i.e., they have the same text and order of the tokens or in other words they are identical), then the answer predicted by the model is considered correct and the exact match score for that instance is 1.

- If the normalized predicted answer does not match the normalized ground truth answer exactly (not identical), then the exact match score for that instance is 0.

Finally, we calculate the overall exact match score for the model by averaging the exact match scores for all the instances in our dataset.

5.3.5 Jaccard Index

Jaccard Index is a popular similarity metric used in natural language processing (NLP) to measure the similarity between two sets of words. It is also known as the Jaccard similarity coefficient or Jaccard coefficient. In question-answering, the Jaccard measures

the similarity between the set of words in the predicted answer and the set of words in the ground truth answer. These are the steps we followed to compute the Jaccard Index for each instance in our dataset:

1. Normalize both the model's predicted answer and the ground truth answer. We previously explained the normalization process in subsection 5.3.1.

2. Tokenize the ground truth and predicted answers in order to convert them into lists of words/tokens.

3. Create sets of words from the ground truth answer's list of tokens and the predicted answer's list of tokens.

4. Find the intersection and the union of the two sets of tokens.

5. Calculate the Jaccard Index, which is the ratio of the size of the intersection (intersection_word_length) of the two sets to the size of their union (union_word_length).

Thus, the formula for calculating Jaccard Index is:

$$Jaccard\_Index = \frac{intersection\_word\_length}{union\_word\_length} \tag{5.6}$$

In order to prevent division by zero, when the sizes of the intersection and union of the two sets (set of the words from the normalized predicted answer and set of the word for the normalized ground truth answer) are both zero, we assign a Jaccard Index value of 1.

Finally, we determine the Jaccard Index of the model by computing the average of Jaccard Indexes that were calculated for every single instance in our evaluation set.

5.4 Experimental Results

5.4.1 Experimental Settings

In all of our experiments on SQuAD [73] dataset, we randomly select 10% of the training set as the evaluation set, since there is no published test set provided for this dataset, and the remaining 90% as the training set.

We train all models with AdamW [54] optimizer using a learning rate of 5×10^{-5} and batch size of 16. We used these default hyper-parameters as they are used by most of the state-of-the-art question-answering models. Additionally, the batch size is set based on the model's size and the capacity of our accessible GPU. We use a cluster of 4 NVIDIA V100 GPUs (Graphical Processing Units) as well as the *Cedar* cluster from *Compute Canada* GPU servers, to conduct all of our experiments. We perform the essential preliminary configurations on these GPU servers before carrying out our experiments. Our choice of the *Cedar* cluster is attributed to its substantial computational resources.

Furthermore, we only used the base version of each transformer-based model, as we examined several different experiments.

We use *F1 score*, *Recall*, *Precision*, *Jaccard Index* and *Exact Match* as performance metrics to evaluate the studied transformer-based models for the task of question-answering in different experiments.

The following subsections present the results of our experiments on SQuAD [73] dataset, which is a popular question-answering benchmark. Specifically, we carried out one set of experiments for different variants of our proposed feature-augmented architecture on all question types. Furthermore, we conducted separate experiments to testify performance of the best variant of the feature-augmented architecture on different question types.

5.4.2 Comparing Transformer-Based Models on SQuAD using Different Variants of Feature-Augmented Architecture 4.2.2 for All Questions Types

Direct Feature Augmentation Architecture 4.2.2

Models	Experiments	F1	Recall	Precision	Jaccard Index	Exact Match
BART-Base	Feature Augmentation	**81.28**	**83.91**	**83.38**	**79.86**	**69.15**
	No Feature Augmentation	80.55	83.29	82.61	79.24	68.49
RoBERTa-Base	Feature Augmentation	**81.23**	**84.17**	83.09	**79.76**	**69.31**
	No Feature Augmentation	81.15	83.11	**83.76**	79.65	68.96
Longformer-Base	Feature Augmentation	**81.12**	83.37	**83.33**	**79.73**	**69.08**
	No Feature Augmentation	80.75	**84.20**	82.04	79.38	68.36
BERT-Base	Feature Augmentation	**79.50**	**83.19**	**80.88**	**77.97**	**67.09**
	No Feature Augmentation	78.76	82.47	80.32	77.27	66.49

Table 5.1: Comparing the performance of different transformer-based models on SQuAD using the *Direct Feature Augmentation Architecture* 4.2.2. This architecture concatenates the extracted features directly with the output of the backbone model.

In this experiment, we analyze various transformer-based models and compare their performance for all question types collectively on SQuAD [73]. Furthermore, an investigation was conducted on the impact of the feature-augmented architecture, which was designed without a neural network, as explained previously in 4.2.2 and named as *Direct Feature Augmentation Architecture*. Table 5.1 illustrates that, generally for most of the performance metrics, BART exhibits the best performance among all models, followed by RoBERTa, Longformer, and BERT. Additionally, the performance metrics indicate that the feature-augmented architecture enhanced the performance of all of the backbone models, except precision score in RoBERTa and recall score in Longformer. Table 5.1 presents a few cases of trade-off between precision and recall score, where the precision score was increased with feature augmentation while the recall score did not (for

Longformer-Base model) or the recall score was increased with feature augmentation but precision score did not (for RoBERTa-Base model). In such cases, since F1 score is a harmonic mean of precision and recall, it can serve as a reliable indicator of the enhanced performance of the backbone model with the help of the feature-augmented architecture.

LSTM Feature Transformation Architecture 4.2.2

Models	Experiments	F1	Recall	Precision	Jaccard Index	Exact Match
Longformer-Base	Feature Augmentation	**81.40**	**84.33**	**82.87**	**80.12**	**69.30**
	No Feature Augmentation	80.75	84.20	82.04	79.38	68.36
RoBERTa-Base	Feature Augmentation	81.05	**84.13**	82.73	**79.84**	68.87
	No Feature Augmentation	**81.15**	83.11	**83.76**	79.65	**68.96**
BART-Base	Feature Augmentation	**80.89**	83.16	**83.13**	**79.67**	**69.00**
	No Feature Augmentation	80.55	**83.29**	82.61	79.24	68.49
BERT-Base	Feature Augmentation	**79.32**	**83.30**	**80.40**	**77.85**	**66.97**
	No Feature Augmentation	78.76	82.47	80.32	77.27	66.49

Table 5.2: Comparing the performance of different transformer-based models on SQuAD using the *LSTM Feature Transformation Architecture* 4.2.2. This architecture concatenates the features that are modified throughout an LSTM layer, with the output of the backbone model.

Table 5.1 displays the outcomes of utilizing a feature-augmented architecture where we directly merged the extracted features with the backbone model's output, without any alteration using a neural network architecture. In contrast, Table 5.2 displays the results of a feature-augmented architecture that involves feeding the extracted features through an LSTM layer and subsequently merging them with the backbone model's output. This variant of the feature-augmented architecture is named as *LSTM Feature Transformation Architecture* and explained previously in 4.2.2. The performance metrics presented in table 5.2, demonstrate that this approach to feature augmentation is reliable and effective in enhancing the performance of the underlying backbone models as well.

However, the degree of enhancement is greater when utilizing the set of features, directly, in their original form (results provided in Table 5.1), rather than making modifications using an LSTM layer (results provided in Table 5.2) in most of the cases. Thus, we can conclude that *Direct Feature Augmentation Architecture* 4.2.2 is more successful in improving the performance of the transformer-based models, compared to *LSTM Feature Transformation Architecture* 4.2.2. Moreover, the table reveals that in this variant of the feature-augmented architecture, Longformer demonstrates the best performance among all models, followed by RoBERTa, BART, and BERT.

Linear Feature Transformation Architecture 4.2.2

Models	Experiments	F1	Recall	Precision	Jaccard Index	Exact Match
RoBERTa-Base	Feature Augmentation	**81.75**	**83.61**	**84.14**	**80.46**	**70.10**
	No Feature Augmentation	81.15	83.11	83.76	79.65	68.96
BART-Base	Feature Augmentation	**80.83**	**83.69**	**82.89**	**79.65**	**68.91**
	No Feature Augmentation	80.55	83.29	82.61	79.24	68.49
Longformer-Base	Feature Augmentation	**80.30**	82.91	**82.47**	78.96	68.22
	No Feature Augmentation	80.75	**84.20**	82.04	**79.38**	**68.36**
BERT-Base	Feature Augmentation	**78.88**	81.42	**81.34**	**77.59**	**66.64**
	No Feature Augmentation	78.76	**82.47**	80.32	77.27	66.49

Table 5.3: Comparing the performance of different transformer-based models on SQuAD using the *Linear Feature Transformation Architecture* 4.2.2. This architecture concatenates the features that are modified throughout a simple linear layer, with the output of the backbone model.

Table 5.3 shows the result of utilizing a feature-augmented architecture where we fed the extracted features through a linear layer, followed by a relu layer, before merging them with the backbone model's output. This variant of the feature-augmented architecture is named as *Linear Feature Transformation Architecture* and described previously in 4.2.2. This experiment is conducted across all types of questions on SQuAD [73]. Experimental

results, provided in Table 5.3, indicate that this approach of feature augmentation yields an improvement in the performance of the underlying backbone models. Nonetheless, the extent of the enhancement achieved by this approach is overally less significant when compared to the other two variants of the linguistic feature component, namely the *Direct Feature Augmentation Architecture* 4.2.2 and the *LSTM Feature Transformation Architecture* 4.2.2.

Comparison Between Different Variants of Feature-augmented Architecture 4.2.2 and Different Transformer-based Models

Through a process of averaging various reported performance metrics for different feature-augmented architecture variants on the SQuAD [73] dataset, we can conclude that the *Direct Feature Augmentation Architecture* 4.2.2 has yielded the most significant improvement in the backbone transformer-based model and the overall feature-augmented architecture performance, with average performance score of 79.02 $\pm$0.75. The second most effective variant is the *LSTM Feature Transformation Architecture* 4.2.2, with the average performance score of 78.92 $\pm$0.79, followed by the the *Linear Feature Transformation Architecture* 4.2.2, with average performance score of 78.74 $\pm$1.04. Thus, we have selected the *Direct Feature Augmentation Architecture*, which is the most successful variant of the proposed feature-augmented architecture, for the remainder of our experiments on the SQuAD [73] dataset.

Moreover, after averaging performance metrics for different transformer-based models on the SQuAD [73] dataset, we have concluded that RoBERTa [52] exhibits the highest performance metrics among all models tested, in both cases of no feature augmentation and different variants of feature augmentation.

5.4.3 Comparing Transformer-Based Models on SQuAD using the *Direct Feature Augmentation Architecture* 4.2.2 for *What* Questions

For this setting, we examine multiple transformer-based models and measure their effectiveness only for *What* questions. To do so, we train each model on the complete training set, using the *Direct Feature Augmentation Architecture* or architecture with no augmentation, and then evaluate their performance on *What* questions from the test set. The results presented in Table 5.4 indicate that BART displays the highest performance out of all the models, followed by Longformer, RoBERTa, and BERT, respectively. As the results show, the feature-augmented architecture was effective for improving the performance of all the models when assessing *What* questions, except for RoBERTa. This can show the importance of the selected features set in answering *What* questions. *What* questions are considered as factoid questions (seeking for a single fact/entity). Linguistic features such as named entity recognition (NER) and part-of-speech (POS) tag can be helpful in answering these types of questions.

Models	Experiments	F1	Recall	Precision	Jaccard Index	Exact Match
BART-Base	Feature Augmentation	**80.66**	**83.74**	**82.46**	**78.98**	**68.35**
	No Feature Augmentation	79.90	82.95	81.63	78.43	67.71
Longformer-Base	Feature Augmentation	**80.59**	83.18	**82.50**	**78.88**	**68.43**
	No Feature Augmentation	80.19	**83.97**	81.22	78.70	67.74
RoBERTa-Base	Feature Augmentation	80.47	**83.63**	82.17	78.66	**68.61**
	No Feature Augmentation	**80.48**	82.55	**83.12**	**78.73**	68.38
BERT-Base	Feature Augmentation	**78.74**	**83.01**	**79.76**	**77.13**	**66.29**
	No Feature Augmentation	78.20	82.33	79.49	76.75	66.09

Table 5.4: Comparing the performance of different transformer-based models on SQuAD's *What* questions using the *Direct Feature Augmentation Architecture* 4.2.2.

5.4.4 Comparing Transformer-Based Models on SQuAD using the *Direct Feature Augmentation Architecture* 4.2.2 for *When* Questions

In this particular setup, we assess multiple transformer-based models and measure their effectiveness solely for *When* questions. We accomplish this by training each model on the complete training set, using the *Direct Feature Augmentation Architecture* or architecture with no augmentation, and then evaluating their performance on *When* questions from the test set. The outcomes shown in Table 5.5 reveal that RoBERTa achieved the highest performance among all models, followed by BART, Longformer, and BERT, respectively. Furthermore, the results indicate that the feature-augmented architecture enhances the performance of all the models, except Longformer, when evaluating *When* questions. This can highlight the significance of the extracted features set for most of the transformer-based models in answering *When* questions. *When* questions are considered as factoid questions as well and linguistic features such as named entity recognition (NER) and part-of-speech (POS) tag can be beneficial in answering these types of questions.

Models	Experiments	F1	Recall	Precision	Jaccard Index	Exact Match
RoBERTa-Base	Feature Augmentation	**88.62**	**89.88**	**89.95**	**86.58**	**79.87**
	No Feature Augmentation	88.14	89.26	89.40	85.35	79.48
BART-Base	Feature Augmentation	**88.52**	89.79	**90.09**	**86.30**	**79.61**
	No Feature Augmentation	88.35	**90.77**	89.03	86.12	79.48
Longformer-Base	Feature Augmentation	87.67	89.20	89.24	85.22	78.05
	No Feature Augmentation	**88.66**	**90.56**	**89.49**	**85.91**	**78.83**
BERT-Base	Feature Augmentation	**87.10**	**89.30**	**87.94**	**83.74**	**77.14**
	No Feature Augmentation	85.01	87.11	86.15	82.26	74.42

Table 5.5: Comparing the performance of different transformer-based models on SQuAD's *When* questions using the *Direct Feature Augmentation Architecture* 4.2.2.

5.4.5 Comparing Transformer-Based Models on SQuAD using the *Direct Feature Augmentation Architecture* 4.2.2 for *Where* Questions

In this configuration, we evaluate multiple transformer-based models and assess their effectiveness in answering only *Where* questions. Our approach involves training each model on the entire training set, using the *Direct Feature Augmentation Architecture* or architecture with no augmentation, and then testing their performance on *Where* questions from the test set. The results, as presented in Table 5.6, indicate that BART outperformed all other models, with RoBERTa, Longformer, and BERT following in descending order. Moreover, the results suggest that the feature-augmented architecture improved the performance of all the models, except for F1, precision and exact match score of RoBERTa as well as precision and exact match score of Longformer in answering *Where* questions. *Where* questions are considered as factoid questions and extracted linguistic features set contains features such as named entity recognition (NER) and part-of-speech (POS) tag, which can be advantageous in answering these types of questions.

Models	Experiments	F1	Recall	Precision	Jaccard Index	Exact Match
BART-Base	Feature Augmentation	**81.98**	**84.71**	**84.93**	**78.88**	**69.40**
	No Feature Augmentation	80.67	83.22	83.57	77.11	66.60
RoBERTa-Base	Feature Augmentation	81.71	**85.66**	83.67	**79.76**	68.80
	No Feature Augmentation	**82.02**	84.25	**85.19**	78.80	**69.00**
Longformer-Base	Feature Augmentation	**80.82**	**83.83**	83.07	**77.50**	67.80
	No Feature Augmentation	80.35	83.14	**83.08**	76.76	**68.8**
BERT-Base	Feature Augmentation	**79.10**	**82.42**	**81.43**	**76.81**	**66.80**
	No Feature Augmentation	77.84	82.41	79.22	74.83	65.20

Table 5.6: Comparing the performance of different transformer-based models on SQuAD's *Where* questions using the *Direct Feature Augmentation Architecture* 4.2.2.

5.4.6 Comparing Transformer-Based Models on SQuAD using the *Direct Feature Augmentation Architecture* 4.2.2 for *Why* Questions

We assess the effectiveness of a few different transformer-based models in answering only *Why* questions in this experiment. Our method involves training each model on the whole training set, using the *Direct Feature Augmentation Architecture* or architecture with no augmentation, and then evaluating their performance on *Why* questions from the test set. The results, as shown in Table 5.7, reveal that RoBERTa outperformed all other models, followed by BART, BERT, and Longformer in that order. Additionally, the outcomes suggest that the *Direct Feature Augmentation Architecture* had more impact on the ability of BART, RoBERTa and Longformer models to answer *Why* questions, while it had less effect on improving the performance of BERT model in this regard. Notwithstanding, in general, the *Direct Feature Augmentation Architecture* has demonstrated the capacity to enhance the efficacy of all models to a greater or lesser extent.

Models	Experiments	F1	Recall	Precision	Jaccard Index	Exact Match
RoBERTa-Base	Feature Augmentation	**70.26**	78.40	**70.37**	76.17	**40.63**
	No Feature Augmentation	70.20	**79.51**	68.59	**77.89**	39.58
BART-Base	Feature Augmentation	**70.30**	**74.69**	**73.24**	**76.05**	**42.19**
	No Feature Augmentation	63.51	71.94	62.84	69.52	31.77
BERT-Base	Feature Augmentation	68.25	72.73	**70.52**	**74.05**	**41.67**
	No Feature Augmentation	**68.92**	**75.60**	69.81	73.55	41.15
Longformer-Base	Feature Augmentation	**66.55**	69.73	**69.08**	**74.64**	**42.19**
	No Feature Augmentation	63.65	**72.80**	63.06	70.45	35.42

Table 5.7: Comparing the performance of different transformer-based models on SQuAD's *Why* questions using the *Direct Feature Augmentation Architecture* 4.2.2.

5.4.7 Comparing Transformer-Based Models on SQuAD using the *Direct Feature Augmentation Architecture* 4.2.2 for *How* Questions

In this experiment, we test several transformer-based models to determine their effectiveness in answering *How* questions. Our methodology involves training each model on the entire training dataset, using the *Direct Feature Augmentation Architecture* or architecture with no augmentation, and then assessing their performance on *How* questions taken from the test dataset. The results presented in Table 5.8 indicate that Longformer was the most successful model, followed by RoBERTa, BART, and BERT, in that order. Moreover, the results suggest that the *Direct Feature Augmentation Architecture* had the most significant impact on the performance of RoBERTa and Longformer in answering *How* questions, whereas it had less effect on improving the performance of BART and BERT in this respect. However, in most cases the *Direct Feature Augmentation Architecture* was able to improve the performance of the studied models.

Models	Experiments	F1	Recall	Precision	Jaccard Index	Exact Match
Longformer-Base	Feature Augmentation	**78.88**	79.69	**84.37**	**80.08**	**62.36**
	No Feature Augmentation	77.79	**80.85**	81.21	78.49	60.85
RoBERTa-Base	Feature Augmentation	**78.53**	**80.42**	**83.75**	**79.20**	**61.22**
	No Feature Augmentation	77.79	78.78	83.23	78.23	59.86
BART-Base	Feature Augmentation	77.92	**78.42**	83.40	**79.27**	62.13
	No Feature Augmentation	**78.04**	78.28	**84.13**	78.82	62.13
BERT-Base	Feature Augmentation	77.38	**79.81**	81.39	77.19	59.18
	No Feature Augmentation	**77.65**	79.19	**82.54**	**77.43**	**60.54**

Table 5.8: Comparing the performance of different transformer-based models on SQuAD's *How* questions using the *Direct Feature Augmentation Architecture* 4.2.2.

5.4.8 Comparing Transformer-Based Models on SQuAD using the *Direct Feature Augmentation Architecture* 4.2.2 for *Who* Questions

For this setting, we evaluate the effectiveness of different transformer-based models in answering *Who* questions specifically. Our approach involves training each model on the complete training dataset, using the *Direct Feature Augmentation Architecture* or architecture with no augmentation, and then measuring their performance on only *Who* questions extracted from the test dataset. The outcomes provided in Table 5.9 indicate that Longformer was the most successful model, followed by RoBERTa, BART, and BERT. Additionally, the results suggest that the *Direct Feature Augmentation Architecture* had the most significant influence on the performance of BART, BERT and Longformer in answering *Who* questions, whereas it had less impact on enhancing the performance of RoBERTa in this aspect. Although the *Direct Feature Augmentation Architecture* exhibited the ability to enhance performance in the majority of studied transformer-based models for *Who* questions.

Models	Experiments	F1	Recall	Precision	Jaccard Index	Exact Match
Longformer-Base	Feature Augmentation	**84.53**	86.13	**86.01**	82.91	**78.26**
	No Feature Augmentation	84.52	**86.98**	85.30	**83.13**	77.72
RoBERTa-Base	Feature Augmentation	84.42	**87.23**	84.77	82.78	**78.34**
	No Feature Augmentation	**84.44**	85.91	**85.70**	**82.82**	78.11
BART-Base	Feature Augmentation	**84.52**	**86.85**	**85.31**	**82.80**	**78.19**
	No Feature Augmentation	83.72	86.49	85.03	82.38	77.48
BERT-Base	Feature Augmentation	**82.58**	85.02	**83.46**	**81.27**	**76.54**
	No Feature Augmentation	81.56	**85.58**	82.03	79.96	74.90

Table 5.9: Comparing the performance of different transformer-based models on SQuAD's *Who* questions using the *Direct Feature Augmentation Architecture* 4.2.2.

5.4.9 Comparing Transformer-Based Models on SQuAD using the *Direct Feature Augmentation Architecture* 4.2.2 for *Whom* Questions

In this particular scenario, we examine the efficacy of different transformer-based models in answering *Whom* questions. Our methodology involves training each model on the entire training dataset, using the *Direct Feature Augmentation Architecture* or architecture with no augmentation, and then evaluating their performance solely on *Whom* questions derived from the test dataset. The performance metrics presented in Table 5.10 demonstrate that BART was the most effective model, followed by BERT, RoBERTa, and Longformer. Furthermore, the results reveal that the *Direct Feature Augmentation Architecture* had the most significant impact on improving the ability of BART model to answer *Whom* questions, whereas it had slightly less effect on improving the performance of BERT and no effect on enhancing the performance of RoBERTa and Longformer in this regard . However, we cannot make a strong claim about the metrics reported in this particular setting due to the insufficient number of *Whom* questions present in both test and entire training datasets (Figure 5.2).

Models	Experiments	F1	Recall	Precision	Jaccard Index	Exact Match
BART-Base	Feature Augmentation	**85.25**	**84.17**	**88.89**	**89.72**	**80.56**
	No Feature Augmentation	80.00	79.17	83.33	85.02	77.78
BERT-Base	Feature Augmentation	**78.50**	77.69	**81.94**	**81.69**	**72.22**
	No Feature Augmentation	76.06	**81.02**	78.56	78.55	69.44
RoBERTa-Base	Feature Augmentation	76.03	75.46	79.17	79.48	72.22
	No Feature Augmentation	**81.28**	**80.46**	**84.72**	**84.46**	**75.00**
Longformer-Base	Feature Augmentation	73.25	72.69	76.39	79.55	69.44
	No Feature Augmentation	**76.77**	**76.16**	**79.17**	**80.18**	**72.22**

Table 5.10: Comparing the performance of different transformer-based models on SQuAD's *Whom* questions using the *Direct Feature Augmentation Architecture* 4.2.2.

5.4.10 Comparing Transformer-Based Models on SQuAD using the *Direct Feature Augmentation Architecture* 4.2.2 for *Whose* Questions

This specific setting involves assessing the effectiveness of various transformer-based models in addressing *Whose* questions. Our approach includes training each model on the complete training set, using the *Direct Feature Augmentation Architecture* or architecture with no augmentation, and then evaluating their performance exclusively on *Whose* questions from the test set. The results, as shown in Table 5.11, indicate that RoBERTa was the most successful model, followed by Longformer, BERT, and BART. Furthermore, the findings demonstrate that the *Direct Feature Augmentation Architecture* had a significant impact on enhancing BERT and Longformer's performance to answer *Whose* questions, but it has minor effect on improving RoBERTa and BART's performance in this regard. It is important to mention that a strong claim cannot be made about the metrics presented in this particular scenario because of the limited number of *Whose* questions present in both test and training sets (Figure 5.2).

Models	Experiments	F1	Recall	Precision	Jaccard Index	Exact Match
RoBERTa-Base	Feature Augmentation	76.17	**81.00**	76.28	75.18	68.00
	No Feature Augmentation	**77.02**	80.00	**77.34**	**76.12**	**70.00**
Longformer-Base	Feature Augmentation	**76.58**	**79.10**	**77.78**	**72.03**	64.00
	No Feature Augmentation	72.22	74.35	74.75	71.14	64.00
BERT-Base	Feature Augmentation	**71.96**	**78.67**	**73.14**	**69.59**	**56.00**
	No Feature Augmentation	66.30	71.00	66.27	62.32	52.00
BART-Base	Feature Augmentation	68.27	**75.67**	68.95	67.26	56.00
	No Feature Augmentation	**72.70**	75.00	**73.35**	**74.77**	**66.00**

Table 5.11: Comparing the performance of different transformer-based models on SQuAD's *Whose* questions using the *Direct Feature Augmentation Architecture* 4.2.2.

5.4.11 Comparing Transformer-Based Models on SQuAD using the *Direct Feature Augmentation Architecture* 4.2.2 for *Which* Questions

In this particular setup, we are examining the efficiency of different transformer-based models in responding to questions that begin with *Which*. Our approach involves training each model on the complete training dataset, using the *Direct Feature Augmentation Architecture* or architecture with no augmentation, and then evaluating their performance solely on *Which* questions from the test dataset. The outcomes of our evaluation, as illustrated in Table 5.12, suggest that BART was the most effective model, followed by RoBERTa, Longformer, and BERT. Moreover, the results reveal that the *Direct Feature Augmentation Architecture* did not have any impact on the ability of RoBERTa to answer *Which* questions, whereas it significantly enhanced BERT's performance and resulted in slightly less improvement in Longformer's and BART's performance in this regard. *Which* questions are categorized as factoid questions and some extracted linguistic features like named entity recognition (NER) and part-of-speech (POS) tag, can be useful in answering these types of questions.

Models	Experiments	F1	Recall	Precision	Jaccard Index	Exact Match
BART-Base	Feature Augmentation	**85.42**	**87.85**	**86.41**	82.78	74.35
	No Feature Augmentation	85.29	87.61	86.05	**82.80**	**75.22**
RoBERTa-Base	Feature Augmentation	85.39	87.57	86.39	83.19	75.46
	No Feature Augmentation	**85.98**	**87.68**	**87.47**	**84.38**	**75.71**
Longformer-Base	Feature Augmentation	**85.20**	87.41	**85.94**	**83.06**	**74.97**
	No Feature Augmentation	84.70	**87.63**	85.24	81.64	73.36
BERT-Base	Feature Augmentation	**84.46**	**87.77**	**84.61**	**81.32**	**73.98**
	No Feature Augmentation	82.52	85.26	82.93	79.50	71.38

Table 5.12: Comparing the performance of different transformer-based models on SQuAD's *Which* questions using the *Direct Feature Augmentation Architecture* 4.2.2.

5.5 Feature Augmentation for Long Passages

5.5.1 Experimental Settings

This research examines the performance of the Longformer model [5] on NLQuAD [82], which is a long question-answering dataset, while focusing specifically on different variations of the proposed feature-augmented architecture. Since the NLQuAD [82] dataset features non-factoid questions with **long** documents, the average sequence length of the documents in this dataset is more than the maximum sequence length that can be processed by some transformer-based models like BERT [19], RoBERTa [52], and BART [50]. Due to this constraint we opted not to assess these models on the NLQuAD [82] dataset and only conduct our experiments on NLQuAD [82] using Longformer [5] model. However, various approaches can be implemented to address this issue, such as segmenting the input sequence into chunks with the length that can be processed by BERT [19], RoBERTa [52] and BART [50].

Throughout the training process, we employed the AdamW [54] optimizer with a learning rate of 5×10^{-5} and a batch size of 8. Once again we used these default hyperparameters as they are used by most of the state-of-the-art question-answering models. We determined the batch size based on the model's considerable size and our GPUs' capacity. Additionally, we utilized only the base version of Longformer [5] for all of our experiments.

5.5.2 Comparing Longformer Model on NLQuAD using Different Variants of Feature-Augmented Architecture 4.2.2 for All Questions Types

Linear Feature Transformation Architecture 4.2.2

This experiment employs the feature-augmented architecture wherein the extracted features are first passed through a linear layer, followed by a relu layer, prior to being integrated with the output of the longformer [5] model. This variant of the architecture is named as *Linear Feature Transformation Architecture* and previously described in 4.2.2. The empirical results, as displayed in Table 5.13, provide evidence that out of the three feature-augmented architecture variants, this particular variant demonstrates the highest level of effectiveness in improving the performance of the backbone model in NLQuAD [82]. By comparing the performance of this variant of the architecture on NLQuAD [82] with the same variant on SQuAD [73], the performance metrics indicate a greater improvement in the backbone model's performance on NLQUAD [82] compared to SQuAD [73].

Model	Experiments	F1	Recall	Precision	Jaccard Index	Exact Match
Longformer-Base	Linear Feature Transformation Architecture	**55.11**	**65.96**	**53.48**	**52.91**	**22.59**
	Direct Feature Augmentation Architecture	51.60	63.90	48.44	50.87	18.73
	Backbone Architecture	50.59	61.06	48.22	50.66	20.75
	LSTM Feature Transformation Architecture	42.79	52.38	41.51	38.86	15.43

Table 5.13: Comparing the performance of different variants of Feature-augmented Architecture, with the Longformer [5] serving as the backbone model, on NLQuAD [82] dataset.

Direct Feature Augmentation Architecture 4.2.2

In contradistinction to the preceding experiment, the current experiment explores the overall efficacy of the longformer [5], as the backbone model for the *Direct Feature Augmentation Architecture*, which was designed without a neural network (described previously in 4.2.2), across all types of questions. Table 5.13 reveals that this variant of feature-augmented architecture improves the performance of the backbone model, however, to a lesser extent compared to the *Linear Feature Augmentation Architecture*. After conducting a comparative analysis of the *Direct Feature Augmentation Architecture* on NLQuAD [82] and the *Direct Feature Augmentation Architecture* on SQuAD [73], it is evident from the results that the improvement in the backbone model's performance is greater on NLQuAD [82] than on SQuAD [73].

LSTM Feature Transformation Architecture 4.2.2

In this experiment, we utilize one of the variants of the feature-augmented architecture that incorporates feeding the extracted features through an LSTM layer, followed by merging them with the output of the longformer [5] model. This variant is referred to as *LSTM Feature Transformation Architecture* and previously explained in 4.2.2. The evaluation metrics depicted in Table 5.13 reveal that this specific variant of feature-augmented architecture fails to improve the performance of the backbone model on NLQUAD [82].

5.6 Ablation Study

We select the *Direct Feature Augmentation Architecture* 4.2.2, which is the best variant of feature-augmented architecture on SQuAD [73], and the best transformer-based model, which is RoBERTa [52], as the backbone model for conducting an ablation study, on SQuAD [73] dataset, to evaluate the impact of individual extracted linguistic features on the performance of the feature-augmented architecture. Ablation study is a scien-

tific methodology that involves selectively disabling or removing specific components or features of a model, system, or experiment to assess their individual impact to the overall performance of the model. In this study, we remove each linguistic feature one at a time to investigate its effect on the performance of the *Direct Feature Augmentation Architecture*.

5.6.1 Absence of Named Entity Recognition (NER)

The results depicted in Table 5.14 evince that despite the removal of named entity recognition (NER) from the extracted linguistic feature set, the *Direct Feature Augmentation Architecture* 4.2.2 still yields performance improvements over the backbone model (the model with no feature augmentation). Nonetheless, the architecture that utilizes the entire feature set exhibits a slightly greater performance boost. Hence, the exclusion of NER has a discernible, albeit not significant, effect on the performance of the feature-augmented architecture. It should be noted that NER is a valuable linguistic feature that can enhance the ability of the transformer-based model to answer factoid questions, since these questions ask about an entity, and SQuAD [73] is a factoid question-answering dataset.

Model	Experiments	F1	Recall	Precision	Jaccard Index	Exact Match
RoBERTa-Base	Direct Feature Augmentation Architecture	**81.23**	**84.17**	83.09	79.76	**69.31**
	Direct Feature Augmentation Architecture No NER	81.22	83.53	83.41	**79.89**	69.06
	No Feature Augmentation	81.15	83.11	**83.76**	79.65	68.96

Table 5.14: Comparing the performance of the *Direct Feature Augmentation Architecture* 4.2.2 in the absence of *Named Entity Recognition*.

5.6.2 Absence of Part-of-Speech Tag (POS)

The results presented in Table 5.15 demonstrate that the omission of part-of-speech (POS) tagging had a deleterious impact on the efficacy of the *Direct Feature Augmen-*

tation Architecture 4.2.2. In other words, the feature-augmented architecture failed to improve the performance of the underlying model after the elimination of POS. This can be because POS is a pivotal linguistic feature in question-answering tasks, as it provides disambiguation of words and furnishes information about sentence structure. Moreover, POS is frequently employed as an input for other features, such as Named Entity Recognition (NER) and Parsing (DEP tree). Therefore, it can be inferred that in the absence POS the linguistic feature set is disadvantageous for the *Direct Feature Augmentation Architecture*.

Model	Experiments	F1	Recall	Precision	Jaccard Index	Exact Match
RoBERTa-Base	Direct Feature Augmentation Architecture	**81.23**	**84.17**	83.09	**79.76**	**69.31**
	No Feature Augmentation	81.15	83.11	**83.76**	79.65	68.96
	Direct Feature Augmentation Architecture No POS	80.84	82.51	83.75	79.63	69.24

Table 5.15: Comparing the performance of the *Direct Feature Augmentation Architecture* 4.2.2 in the absence of *Part-of-Speech Tag*.

5.6.3 Absence of Syntactic Dependency (DEP)

The results provided in Table 5.16 illustrate that the exclusion of the syntactic dependency (DEP) feature engenders an unfavorable impact on the efficacy of the *Direct Feature Augmentation Architecture* 4.2.2. Syntactic dependencies furnish valuable insight into the grammatical structure of a sentence and can be utilized to comprehend the associations between words, identify the principal subjects and objects in a sentence, and ascertain the role of each word within a sentence. This information can be advantageous in the context of question-answering. Consequently, in the absence of the DEP feature, the *Direct Feature Augmentation Architecture* is incapable of enhancing the performance of the backbone model.

Model	Experiments	F1	Recall	Precision	Jaccard Index	Exact Match
RoBERTa-Base	Direct Feature Augmentation Architecture	**81.23**	**84.17**	83.09	**79.76**	**69.31**
	No Feature Augmentation	81.15	83.11	**83.76**	79.65	68.96
	Direct Feature Augmentation Architecture No DEP	80.92	84.08	82.53	79.49	68.83

Table 5.16: Comparing the performance of the *Direct Feature Augmentation Architecture* 4.2.2 in the absence of *Syntactic Dependency*.

5.6.4 Absence of Stop Words (STOP)

The provided results in Table 5.17 demonstrate that the *Direct Feature Augmentation Architecture* 4.2.2 continues to yield performance improvements even after the exclusion of stop words (STOP) from the extracted linguistic feature set. Specifically, the results suggest that the *Direct Feature Augmentation Architecture* 4.2.2 without stop words may outperform the *Direct Feature Augmentation Architecture* that uses all extracted linguistic features, in some cases. This can show the success of the transformer-based backbone model (which is RoBERTa [52] in this study) in capturing the stop words. Stop words carry minimal semantic content and contribute little value to question-answering tasks. Transformer-based models can learn to allocate less attention towards them as they are unlikely to form a part of the answer.

Model	Experiments	F1	Recall	Precision	Jaccard Index	Exact Match
RoBERTa-Base	Direct Feature Augmentation Architecture No STOP	**81.41**	83.81	83.53	**79.88**	69.28
	Direct Feature Augmentation Architecture	81.23	**84.17**	83.09	79.76	**69.31**
	No Feature Augmentation	81.15	83.11	**83.76**	79.65	68.96

Table 5.17: Comparing the performance of the *Direct Feature Augmentation Architecture* 4.2.2 in the absence of *Stop Words*.

5.7 Case Study on SQuAD

We conduct a few case studies on the SQuAD [73] dataset, with the objective of demonstrating how specifically certain linguistic features contribute to the performance of a backbone transformer-based model in identifying answer spans within context passages for given questions. The *Direct Feature Augmentation Architecture* is utilized along with RoBERTa [52] as the backbone transformer-based model to showcase the observed improvements in performance.

Question: In which era was Frédéric leave a legacy of as a leading symbol?

Passage: In his native Poland, in France, where he composed most of his works, and beyond, Chopin's music, his status as one of music's earliest superstars, his association (if only indirect) with political insurrection, his love life and his early death have made him, in the public consciousness, a leading symbol of the **Romantic era**. His works remain popular, and he has been the subject of numerous films and biographies of varying degrees of historical accuracy.

Predicted Answer by Feature Augmentation: Romantic era

Predicted Answer by No Feature Augmentation: Romantic

Figure 5.4: An example of question-passage pair from SQuAD [73], which showcases the importance of *Syntactic Dependency (DEP)* linguistic feature. The ground truth answer is bolded within the passage.

5.7.1 Effectiveness of Syntactic Dependency (DEP)

The illustration presented in Figure 5.4 exemplifies a scenario where the transformer-based model, serving as the backbone architecture without any feature augmentation, falls short in identifying a comprehensive and complete answer to a given query. In contrast, the *Direct Feature Augmentation Architecture* effectively locates the complete and accurate answer.

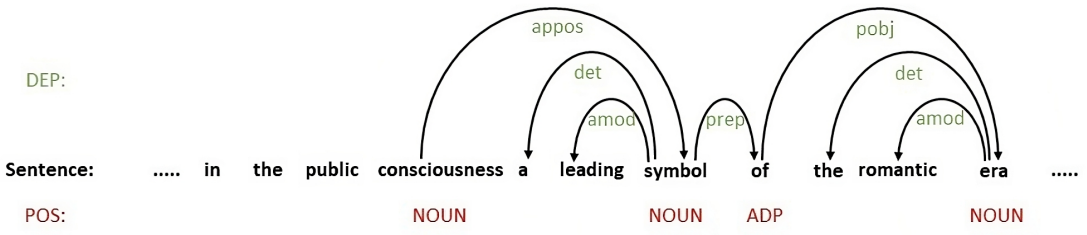


Figure 5.5: Dependency tree constructed for the sentence within the passage, which contains the answer to the given question, in the example provided by Figure 5.4. The red labels signify the POS attributes of individual words while the green labels correspond to the DEP features.

Figure 5.5 depicts the syntactic relationships among the words within the sentence that encompasses the ground truth answer, from the example demonstrated in Figure 5.4. The relationship between the terms *of* and *era* is classified as **pobj** or object of preposition, which is a nominal located in the object position. Additionally, the syntactic dependency between the words *era* and *romantic* is denoted as **amod** or adjectival modifier, which represents an adjectival phrase that serves to modify the noun (or pronoun). These syntactic dependencies enable the **Direct Feature Augmentation Architecture** to locate the entire and precise answer to the given question, which is searching for an object for the phrase *symbol of*.

Question: On what date was Frédéric born on?

Passage: Fryderyk Chopin was born in Żelazowa Wola, 46 kilometres (29 miles) west of Warsaw, in what was then the Duchy of Warsaw, a Polish state established by Napoleon. The parish baptismal record gives his birthday as **22 February 1810**, and cites his given names in the Latin form Fridericus Franciscus (in Polish, he was Fryderyk Franciszek). However, the composer and his family used the birthdate 1 March,[n 2] which is now generally accepted as the correct date.

Predicted Answer by Feature Augmentation: 22 February 1810

Predicted Answer by No Feature Augmentation: 22 February 1810, and cites his given names in the Latin form Fridericus Franciscus (in Polish, he was Fryderyk Franciszek). However, the composer and his family used the birthdate 1 March,[

Figure 5.6: An example of question-passage pair from SQuAD [73], which showcases the importance of *Named Entity Recognition (NER)* and *Syntactic Dependency (DEP)* linguistic features. The ground truth answer is bolded within the passage.

5.7.2 Simultaneous Effectiveness of Named Entity Recognition (NER) and Syntactic Dependency (DEP)

Figure 5.6 presents an example where a transformer-based model, with no feature augmentation, fails to accurately identify the exact answer to a given question and instead produces extraneous unnecessary information. Despite, the *Direct Feature Augmentation Architecture*, employs named entity recognition and syntactic dependency analysis to locate the answer. Specifically, *DATE* is mapped to *February 1810* as the named entity

feature and **nummod** or numeric modifier, which is any number phrase that serves to modify the meaning of the noun with a quantity, is the dependency between *February* and *22*, as well as between *February* and *1810*. These linguistic features allow the *Direct Feature Augmentation Architecture* to consider **22 February 1810** as a single phrase for the date of birth, thereby accurately identifying the answer.

5.8 Summary

In this chapter, we presented an assessment of the effectiveness of the proposed feature-augmented architecture, along with all of the designed variants of this architecture, for the task of answer span detection in response to a given question, across two datasets: SQuAD [73] and NLQuAD [82]. Our primary emphasis is on SQuAD [73] and most of our experiments were conducted on this dataset. Moreover, we only utilized longformer [5] model to conduct a few experiments on NLQuAD [82] given the challenges posed by this dataset in processing lengthy text. We conducted several experiments on different variations of feature-augmented architecture with different transformer-based models on SQuAD [73]. Additionally, we examined the performance of each transformer-based model, with feature augmentation and without feature augmentation, on extracting answers from a given passage for different types of questions provided in SQuAD [73] dataset.

The reported performance metrics on SQuAD [73] indicate that, in the majority of our experiments, either RoBERTa [52], BART [50] or Longformer [5] outperform BERT [19] model. This can highlight the significance of using larger mini-batches and more data during training, as well as employing dynamic masked language modeling as a pre-training task to improve language understanding in RoBERTa [52]. Additionally, it can underscore the importance of BART's [50] usage of various transformation functions during pre-training for the masked language modeling unsupervised task to alter the original

input text. Moreover, this can denote the proficiency of the longformer [5] architecture in achieving favorable performance outcomes while trading off some information by only attending to a restricted scope of information to reduce memory consumption.

Furthermore, when comparing the performance of each backbone model with its respective feature-augmented architecture, in different variations of the architecture, the evaluation metrics reveal that the suggested variants of feature-augmented architectures are generally effective in enhancing the performance of the backbone models. Notably, for SQuAD [73] dataset, the *Direct Feature Augmentation Architecture* 4.2.2 demonstrate greater consistency and success in improving the backbone models' performance followed by the *LSTM Feature Transformation Architecture* 4.2.2 followed by *Linear Feature Transformation Architecture* 4.2.2. While, for NLQuAD [82] dataset, the *Linear Feature Transformation Architecture* 4.2.2 is more successful in improving the performance of Longformer [5] compared to the *Direct Feature Augmentation Architecture* 4.2.2. However, the *LSTM Feature Transformation Architecture* 4.2.2 does not yield any improvement in the performance of the longformer model [5] on NLQuAD [82].

In addition, experiments on SQuAD [73] dataset, suggest that the *Direct Feature Augmentation Architecture* 4.2.2 has been found to enhance the performance of the transformer-based backbone model more significantly for certain types of questions compared to the others. This disparity in performance improvement could be attributed to the varying sizes of each question type's category in the training and test sets. Furthermore, it can be due to the importance of the chosen linguistic features set for some specific types of questions, such as *What*, *Where* or *When* questions, more than the others.

Moreover, the ablation study conducted on the SQuAD [73] dataset reveals that the exclusion of either part-of-speech (POS) tags or syntactic dependencies (DEP) from the feature set adversely affects the performance of the *Direct Feature Augmentation Architecture*. Conversely, the removal of stop words (STOP) from the feature set can have a positive impact on the performance of the aforementioned architecture. Furthermore,

the omission of named entity recognition (NER) from the feature set does not significantly impact the performance of the *Direct Feature Augmentation Architecture*. Even in the absence of NER, this architecture can enhance the performance of the transformer-based backbone model, albeit to a lesser degree than when NER is included in the feature set.

Ultimately, through an analysis of a few case studies taken from the SQuAD [73] dataset, we have demonstrated the potential benefits of employing particular extracted linguistic features in facilitating the process of locating the answer within the context to the given question.

Chapter 6

Conclusion and Future work

This research aimed to study the performance of different transformer-based models in the task of answer span detection or extractive question-answering.

We proposed a feature-augmented architecture, along with three variants, to enhance the performance of the studied potent transformer-based language models in question-answering. Furthermore, a comparative analysis was conducted among the investigated models to determine the most efficacious transformer-based models on the SQuAD [73] dataset, a comprehensive dataset designed for extractive question-answering task. Additionally, the impact of the proposed feature-augmented architectures on improving the backbone transformer-based model's performance was examined on SQuAD [73]. However, we only studied the effect of feature-augmented architecture for Longformer [5] model on NLQUAD [82], with the aim of exploring a non-factoid long question-answering dataset as well. Moreover, a series of experiments were carried out to assess the effectiveness of each transformer-based model and feature-augmented transformer-based model in answering specific types of questions on SQuAD [73] dataset.

Furthermore, an ablation study was carried out on the SQuAD [73] dataset, in order to investigate the effect of each specific feature from the extracted set of features, independently, on the efficacy of the most successful variant of the feature-augmented

architecture.

6.1 Thesis Contribution Highlights

The main contribution of this thesis in Chapter 4 and Chapter 5 can be summarized as follows:

- **Feature-augmented Transformer-based Architecture**

  This study presents a feature-augmented architecture that incorporates linguistic token-level features to enhance the performance of transformer-based models in the task of question-answering. Although transformer-based language models have demonstrated remarkable success across a variety of natural language processing (NLP) tasks due to their high-level language understanding, they can face challenges in analyzing complex linguistic structures present in certain questions. Specifically, questions with intricate logic and clause structures can pose difficulties for some transformer-based models. We propose three different variations of the transformer-based feature-augmented architecture in this research, which can be helpful in addressing this issue. The results indicate that incorporating additional linguistic features can improve the performance of the backbone transformer-based model in most cases. These findings suggest that employment of off-the-shelf linguistic features can aid the backbone model in better understanding complex context passages and consequently improve the accuracy of the question-answering transformer-based model.

- **Ablation Study Demonstrating the Effect of Individual Features on the Feature-augmented Architecture**

  An ablation study was conducted to evaluate the effectiveness of individual features on the most successful variant of feature-augmented architecture, referred to as *Direct Feature Augmentation Architecture*. The results indicate that the absence

of *Syntactic Dependency (DEP)* and *Part-of-Speech (POS) Tag* from the features set had the greatest negative impact on the performance of the *Direct Feature Augmentation Architecture*.

- **Case Studies Demonstrating the Importance of Some Linguistic Features** This research presents a few examples, taken from SQuAD [73] dataset, to illustrate the limitations of the backbone transformer-based model, as well as the superior performance of the proposed feature-augmented transformer-based model in locating the exact accurate answer to the given question within thhe given context.

6.2 Limitations

The experimental results demonstrated that different variants of proposed feature-augmented architecture can improve the performance of the transformer-based backbone model (with no feature augmentation) in most cases. However, there are some limitations accompanied within the conducted experiments as well as the proposed variants of the architecture, which should be considered.

- The experiments conducted on NLQuAD [82] do not include the examination of the performance of BERT [19], RoBERTa [52], and BART [50] models, as the maximum sequence length accommodated by these models is substantially less than the average length of the context passages in NLQuAD [82].

- Due to the need to evaluate each variant for four distinct transformer-based models, only a limited number of feature-augmented transformer-based architecture variations were proposed and examined.

- The proposed feature-augmented transformer-based architecture for the task of question-answering utilized only a restricted selection of linguistic token-level fea-

tures, which were extracted using of-the-shelf linguistic features extraction tools, and incorporated into the architecture.

- The scope of this study was limited to the evaluation and comparison of four transformer-based models for the task of question-answering. All of the selected models expect the similar input and output formats.

- The ablation study was carried out only on the transformer-based model with the best performance and across *all* question types, not *different* question types, on SQuAD [73].

6.3 Future Work

The proposed feature-augmented transformer-based architecture for the task of answer span detection in this thesis presents a promising avenue for future work. Given that the performance of some transformer-based models on NLQuAD [82] has yet to be explored, we recommend incorporating those models by addressing the sequence length limitation through the segmentation of context passages and searching for the answer in each segment. This approach allows each segment to contain a portion or the entirety of the answer. Furthermore, it may be beneficial to expand the linguistic token-level feature set to examine the impact of additional features that could aid in answering specific question types, such as the use of cue words like *because* to answer *why* questions, or usage of more features in general for *all* question types. Additionally, the proposed transformer-based feature-augmented architecture for the task of question-answering is designed in a way to accommodate the substitution of any neural network to modify the extracted feature set. Further investigation could also consider evaluating other transformer-based models, including T5 [72]. T5 differs from the studied transformer-based models in this thesis, as it is a multi-task learning model that utilizes natural language prefixes and trains the architecture in a unified objective manner, modeling all problems in a text-to-text format

using the encoder-decoder format proposed by transformers [91]. Moreover, an ablation study could be conducted to see the importance and effectiveness of each linguistic feature on different question types for different transformer-based models.

Bibliography

[1] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics.

[2] Ali Mohamed Nabil Allam and Mohamed Hassan Haggag. The question answering systems: A survey. *International Journal of Research and Reviews in Information Sciences (IJRRIS)*, 2(3), 2012.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[5] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *CoRR*, abs/2004.05150, 2020.

[6] Jonathan Berant, Vivek Srikumar, Pei-Chun Chen, Abby Vander Linden, Brittany Harding, Brad Huang, Peter Clark, and Christopher D. Manning. Modeling biological processes for reading comprehension. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1499–1510, Doha, Qatar, October 2014. Association for Computational Linguistics.

[7] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. *CoRR*, abs/1508.05326, 2015.

[8] Daniel Braun, Adrian Hernandez Mendez, Florian Matthes, and Manfred Langen. Evaluating natural language understanding services for conversational question answering systems. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 174–185, Saarbrücken, Germany, August 2017. Association for Computational Linguistics.

[9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[10] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading Wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[11] Long Chen, Yuhang Zheng, Yulei Niu, Hanwang Zhang, and Jun Xiao. Counterfactual samples synthesizing and training for robust visual question answering. *CoRR*, abs/2110.01013, 2021.

[12] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *CoRR*, abs/1601.06733, 2016.

[13] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *CoRR*, abs/1904.10509, 2019.

[14] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[15] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[16] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. *CoRR*, abs/1705.02364, 2017.

[17] Andrew M. Dai and Quoc V. Le. Semi-supervised sequence learning. *CoRR*, abs/1511.01432, 2015.

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

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[20] William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

[21] Cícero dos Santos, Luciano Barbosa, Dasha Bogdanova, and Bianca Zadrozny. Learning hybrid representations to retrieve semantically equivalent questions. In

Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers), pages 694–699, Beijing, China, July 2015. Association for Computational Linguistics.

[22] Abdessamad Echihabi and Daniel Marcu. A noisy-channel approach to question answering. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 16–23, Sapporo, Japan, July 2003. Association for Computational Linguistics.

[23] Angela Fan, Yacine Jernite, Ethan Perez, David Grangier, Jason Weston, and Michael Auli. ELI5: long form question answering. *CoRR*, abs/1907.09190, 2019.

[24] Minwei Feng, Bing Xiang, Michael R. Glass, Lidan Wang, and Bowen Zhou. Applying deep learning to answer selection: A study and an open task. *CoRR*, abs/1508.01585, 2015.

[25] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *CoRR*, abs/1705.03122, 2017.

[26] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005. IJCNN 2005.

[27] Scott Gray, Alec Radford, and Diederik P. Kingma. Gpu kernels for block-sparse weights. 2017.

[28] Helia Hashemi, Mohammad Aliannejadi, Hamed Zamani, and W. Bruce Croft. ANTIQUE: A non-factoid question answering benchmark. *CoRR*, abs/1905.08957, 2019.

[29] Hua He and Jimmy Lin. Pairwise word interaction modeling with deep neural networks for semantic similarity measurement. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 937–948, San Diego, California, June 2016. Association for Computational Linguistics.

[30] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

[31] Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016.

[32] Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *CoRR*, abs/1506.03340, 2015.

[33] Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. The goldilocks principle: Reading children's books with explicit memory representations. *CoRR*, abs/1511.02301, 2015.

[34] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[35] Jeremy Howard and Sebastian Ruder. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146, 2018.

[36] Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. Knowledge graph embedding based question answering. In *ACM International Conference on Web Search and Data Mining*, 2019.

[37] Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. *CoRR*, abs/2007.01282, 2020.

[38] Yacine Jernite, Samuel R. Bowman, and David A. Sontag. Discourse-based objectives for fast unsupervised sentence representation learning. *CoRR*, abs/1705.00557, 2017.

[39] Zhengbao Jiang, Jun Araki, Haibo Ding, and Graham Neubig. Understanding and improving zero-shot multi-hop reasoning in generative question answering, 2022.

[40] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. Spanbert: Improving pre-training by representing and predicting spans. *CoRR*, abs/1907.10529, 2019.

[41] Mandar Joshi, Omer Levy, Daniel S. Weld, and Luke Zettlemoyer. BERT for coreference resolution: Baselines and analysis. *CoRR*, abs/1908.09091, 2019.

[42] Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *CoRR*, abs/1602.02410, 2016.

[43] Hasangi Kahaduwa, Dilshan Pathirana, Pathum Liyana Arachchi, Vishma Dias, Surangika Ranathunga, and Upali Sathyajith Kohomban. Question answering system for the travel domain. *2017 Moratuwa Engineering Research Conference (MERCon)*, pages 449–454, 2017.

[44] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online, November 2020. Association for Computational Linguistics.

[45] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. Structured attention networks. *CoRR*, abs/1702.00887, 2017.

[46] Tomás Kociský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The narrativeqa reading comprehension challenge. *CoRR*, abs/1712.07040, 2017.

[47] Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. Revealing the dark secrets of BERT. *CoRR*, abs/1908.08593, 2019.

[48] Oleksii Kuchaiev and Boris Ginsburg. Factorization tricks for LSTM networks. *CoRR*, abs/1703.10722, 2017.

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

[50] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461, 2019.

[51] Zhouhan Lin, Minwei Feng, Cícero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *CoRR*, abs/1703.03130, 2017.

[52] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[53] Lajanugen Logeswaran and Honglak Lee. An efficient framework for learning sentence representations. *CoRR*, abs/1803.02893, 2018.

[54] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.

[55] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015.

[56] Shangwen Lv, Daya Guo, Jingjing Xu, Duyu Tang, Nan Duan, Ming Gong, Linjun Shou, Daxin Jiang, Guihong Cao, and Songlin Hu. Graph-based reasoning over heterogeneous external knowledge for commonsense question answering. *CoRR*, abs/1909.05311, 2019.

[57] Christopher D. Manning. Last words: Computational linguistics and deep learning. *Computational Linguistics*, 41(4):701–707, December 2015.

[58] Ryan Musa, Xiaoyan Wang, Achille Fokoue, Nicholas Mattei, Maria Chang, Pavan Kapanipathi, Bassem Makni, Kartik Talamadupula, and Michael Witbrock. Answering science exam questions using query rewriting with background knowledge. *CoRR*, abs/1809.05726, 2018.

[59] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. MS MARCO: A human generated machine reading comprehension dataset. *CoRR*, abs/1611.09268, 2016.

[60] Jianmo Ni, Chenguang Zhu, Weizhu Chen, and Julian J. McAuley. Learning to attend on essential terms: An enhanced retriever-reader model for scientific question answering. *CoRR*, abs/1808.09492, 2018.

[61] Bogdan-Ioan Nicula, Stefan Ruseti, and Traian Rebedea. Improving deep learning for multiple choice question answering with candidate contexts. In *European Conference on Information Retrieval*, 2018.

[62] Rodrigo Frassetto Nogueira and Kyunghyun Cho. Task-oriented query reformulation with reinforcement learning. *CoRR*, abs/1704.04572, 2017.

[63] Hariom A. Pandya and Brijesh S. Bhatt. Question answering survey: Directions, challenges, datasets, evaluation matrices. *CoRR*, abs/2112.03572, 2021.

[64] Ankur P. Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. *CoRR*, abs/1606.01933, 2016.

[65] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *CoRR*, abs/1705.04304, 2017.

[66] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[67] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018.

[68] Chen Qu, Liu Yang, Cen Chen, Minghui Qiu, W. Bruce Croft, and Mohit Iyyer. Open-retrieval conversational question answering. *CoRR*, abs/2005.11364, 2020.

[69] Chen Qu, Liu Yang, Minghui Qiu, W. Bruce Croft, Yongfeng Zhang, and Mohit Iyyer. Bert with history answer embedding for conversational question answering. In *SIGIR 2019 - Proceedings of the 42nd International ACM SIGIR Conference*

on Research and Development in Information Retrieval, SIGIR 2019 - Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 1133–1136. Association for Computing Machinery, Inc, July 2019. 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019 ; Conference date: 21-07-2019 Through 25-07-2019.

[70] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

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

[72] 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. *CoRR*, abs/1910.10683, 2019.

[73] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics.

[74] Mengye Ren, Ryan Kiros, and Richard S. Zemel. Image question answering: A visual semantic embedding model and a new dataset. *CoRR*, abs/1505.02074, 2015.

[75] Matthew Richardson, Christopher J.C. Burges, and Erin Renshaw. MCTest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 193–203, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[76] Ellen Riloff and Michael Thelen. A rule-based question answering system for reading comprehension tests. In *ANLP-NAACL 2000 Workshop: Reading Comprehension Tests as Evaluation for Computer-Based Language Understanding Systems*, 2000.

[77] Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *CoRR*, abs/1509.00685, 2015.

[78] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *CoRR*, cs.CL/0306050, 2003.

[79] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.

[80] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *CoRR*, abs/1701.06538, 2017.

[81] Sonit Singh. Natural language processing for information extraction. *CoRR*, abs/1807.02383, 2018.

[82] Amir Soleimani, Christof Monz, and Marcel Worring. NLQuAD: A non-factoid long question answering data set. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1245–1255, Online, April 2021. Association for Computational Linguistics.

[83] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. Weakly supervised memory networks. *CoRR*, abs/1503.08895, 2015.

[84] Mihai Surdeanu, Massimiliano Ciaramita, and Hugo Zaragoza. Learning to rank answers to non-factoid questions from web collections. *Computational Linguistics*, 37(2):351–383, June 2011.

[85] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[86] Ming Tan, Bing Xiang, and Bowen Zhou. Lstm-based deep learning models for non-factoid answer selection. *CoRR*, abs/1511.04108, 2015.

[87] Wilson L. Taylor. "cloze procedure": A new tool for measuring readability. *Journalism & Mass Communication Quarterly*, 30:415 – 433, 1953.

[88] Hrishikesh Terdalkar and Arnab Bhattacharya. Framework for question-answering in Sanskrit through automated construction of knowledge graphs. In *Proceedings of the 6th International Sanskrit Computational Linguistics Symposium*, pages 97–116, IIT Kharagpur, India, October 2019. Association for Computational Linguistics.

[89] Joseph Turian, Lev-Arie Ratinov, and Yoshua Bengio. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

[90] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.

[91] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[92] Di Wang and Eric Nyberg. A long short-term memory model for answer sentence selection in question answering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 707–712, Beijing, China, July 2015. Association for Computational Linguistics.

[93] Luqi Wang, Kaiwen Zheng, Liyin Qian, and Sheng Li. A survey of extractive question answering. In *2022 International Conference on High Performance Big Data and Intelligent Systems (HDIS)*, pages 147–153, 2022.

[94] Shuohang Wang and Jing Jiang. A compare-aggregate model for matching text sequences. *CoRR*, abs/1611.01747, 2016.

[95] Zhiguo Wang, Wael Hamza, and Radu Florian. Bilateral multi-perspective matching for natural language sentences. *CoRR*, abs/1702.03814, 2017.

[96] Adina Williams, Nikita Nangia, and Samuel R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *CoRR*, abs/1704.05426, 2017.

[97] Felix Wu, Angela Fan, Alexei Baevski, Yann N. Dauphin, and Michael Auli. Pay less attention with lightweight and dynamic convolutions. *CoRR*, abs/1901.10430, 2019.

[98] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural ma-

chine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[99] Vikas Yadav, Steven Bethard, and Mihai Surdeanu. Alignment over heterogeneous embeddings for question answering. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2681–2691, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[100] Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. End-to-end open-domain question answering with bertserini. *CoRR*, abs/1902.01718, 2019.

[101] Yi Yang, Wen-tau Yih, and Christopher Meek. WikiQA: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2013–2018, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[102] Zekun Yang, Noa Garcia, Chenhui Chu, Mayu Otani, Yuta Nakashima, and Haruo Takemura. Bert representations for video question answering. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

[103] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237, 2019.

[104] Lei Yu, Karl Moritz Hermann, Phil Blunsom, and Stephen Pulman. Deep learning for answer sentence selection. *CoRR*, abs/1412.1632, 2014.

[105] Yue Zhang. The death of feature engineering? —— bert with liguistic features on squad 2.0. 2019.

[106] Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander Smola, and Le Song. Variational reasoning for question answering with knowledge graph. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018.

[107] Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *ICCV*, pages 19–27. IEEE Computer Society, 2015.