



Aplicatii NLP pentru dialogurile cu un robot orientate catre sarcini

Constantin Nicolae

Coordinator: Academician **Dan TUFIS**

ROMANIAN ACADEMY

RESEARCH INSTITUTE FOR ARTIFICIAL INTELLIGENCE

"MIHAI DRAGANESCU"

2022

TABLE OF CONTENTS

| | |
|---|-----------|
| List of Tables | 4 |
| List of Figures | 5 |
| Chapter 1: Introduction | 1 |
| Chapter 2: Motivation | 4 |
| 2.1 Problem Definition | 4 |
| 2.2 Question Answering Procedure | 9 |
| 2.3 Challenges of the Question Answering system | 11 |
| Chapter 3: Dataset | 13 |
| 3.1 Introduction | 13 |
| 3.2 Various QA datasets | 14 |
| 3.2.1 Stanford Question Answering Dataset (SQuAD) | 14 |
| 3.2.2 QUASAR | 15 |
| 3.2.3 TriviaQA | 17 |
| 3.2.4 MS-MARCO | 17 |
| 3.2.5 RACE | 18 |
| 3.2.6 MS MACRO | 18 |
| 3.2.7 Narrative QA | 19 |
| 3.2.8 MCScript Dataset | 19 |
| 3.3 Datasets Comparison | 20 |
| 3.4 Open-Domain Question Answering | 22 |
| 3.4.1 DrQA | 22 |
| 3.4.2 Wikipedia QA | 22 |
| 3.5 Romanian Dataset | 23 |
| 3.5.1 Romanian QA Task | 23 |
| 3.5.2 Romanian Question Classification Task | 28 |
| 3.5.3 Visual Analysis of the Dataset | 30 |
| 3.6 Evaluation | 33 |
| 3.6.1 Support Vector Machine | 34 |
| 3.6.2 Naive Bayes | 34 |

| | | |
|---|---|-----------|
| 3.6.3 | Decision Tree | 35 |
| 3.6.4 | Tsetlin Machine | 35 |
| 3.6.5 | Results | 37 |
| 3.6.6 | Discussion | 37 |
| 3.7 | Outlier Detection | 42 |
| 3.7.1 | VAE-GAN Based Zero-shot Outlier Detection | 44 |
| 3.7.2 | Results ROITD outlier detection | 48 |
| Chapter 4: Machine learning advancements | | 52 |
| 4.1 | Machine Learning | 52 |
| 4.2 | Deep Learning | 52 |
| 4.2.1 | Neural Networks | 53 |
| 4.2.2 | Recurrent Neural Networks | 53 |
| 4.2.3 | Long-Short Term Memory | 54 |
| 4.3 | Natural Language Processing | 55 |
| 4.3.1 | Language Modeling | 55 |
| 4.3.2 | Encoder-Decoder Model | 56 |
| 4.3.3 | NLP Tasks | 56 |
| 4.4 | Transfer Learning | 58 |
| 4.4.1 | Fine-tuning | 58 |
| 4.4.2 | Domain Adaptation | 59 |
| 4.5 | BERT | 60 |
| 4.5.1 | Attention | 60 |
| 4.5.2 | Transformers | 61 |
| 4.5.3 | Model Pre-training | 64 |
| 4.5.4 | Model Fine-Tuning | 64 |
| 4.6 | ALBERT | 65 |
| 4.7 | Acoustics Model Advancements | 70 |
| 4.8 | Introduction to Bayesian Inference | 88 |
| Chapter 5: Proposed QA Methodology | | 95 |
| 5.1 | Related Pipelines | 95 |
| 5.2 | Implementation in Google Cloud Service | 101 |
| 5.2.1 | Implementation in Webpage | 102 |
| 5.2.2 | Implementation in Pepper | 102 |

| | | |
|---|---|------------|
| 5.2.3 | Integration Dailogflow: Pepper | 106 |
| 5.3 | Haystack | 111 |
| 5.3.1 | File Conversion | 114 |
| 5.3.2 | Preprocessing | 115 |
| 5.3.3 | DocumentStores | 115 |
| 5.3.4 | Elastic Search | 115 |
| 5.3.5 | Retrievers | 116 |
| 5.3.6 | Readers | 117 |
| 5.3.7 | Pipelines | 117 |
| 5.3.8 | Results | 117 |
| 5.4 | QA using Knowledge Graphs | 122 |
| 5.4.1 | Results | 128 |
| Chapter 6: QA Using Bayesian Inference | | 132 |
| 6.1 | Next best Answer using Bayesian Inference | 133 |
| Chapter 7: Conclusions | | 138 |
| 7.1 | Conclusions | 138 |
| 7.2 | Next steps | 142 |
| 7.2.1 | Entailment Trees | 142 |
| 7.2.2 | Extra-Linguistic Contexts | 144 |
| 7.2.3 | Neural Natural Logic Inference | 146 |
| Bibliography | | 148 |

LIST OF TABLES

| | | |
|-----|--|-----|
| 3.1 | Categorization of 50 randomly chosen Question Answer pairs based on reasoning. [149] | 25 |
| 3.2 | Types of unanswerable RoITD questions [149] | 26 |
| 3.3 | RoITD question length statistics. [149] | 26 |
| 3.4 | RoITD answer length statistics. [149] | 26 |
| 3.5 | RoITD passage length statistics. [149] | 26 |
| 3.6 | Statistics of dataset for different types of questions. | 29 |
| 3.7 | Performance of various models on question classification task. . . . | 37 |
| 3.8 | Performance of different outlier detection algorithms on RoITD dataset | 50 |
| 5.1 | Dataset Statistics for splitted train and test sets. | 117 |
| 5.2 | RoITD dataset baseline performance. | 117 |
| 5.3 | Results for the EM and F1 metrics | 131 |
| 6.1 | An Illustration of the problem. | 133 |
| 6.2 | Effect of a single Bayesian update on the accuracy of the 1st answer. 136 | |
| 6.3 | Effect of a single Bayesian update on the 2nd answer if 1st answer is rejected. | 136 |

LIST OF FIGURES

| | | |
|------|--|----|
| 2.1 | Sample DL RC-QA model architecture. [25] | 6 |
| 2.2 | Summary of recently reported KB-QA methods. [25] | 6 |
| 2.3 | Summary of KB-QA benchmark datasets that involve complex ques- tions. [76] | 7 |
| 2.4 | Question mapping templates in Aquu including example candidates and corresponding questions. [31, 76] | 10 |
| 3.1 | A sample from SQuAD. | 15 |
| 3.2 | Sample problem instance from QUASAR-S. | 16 |
| 3.3 | Sample problem instance from QUASAR-T. | 16 |
| 3.4 | Sample problem instance from TriviaQA. | 17 |
| 3.5 | Sample problem instance from MS-MARCO. | 18 |
| 3.6 | Sample problem instance from RACE. | 19 |
| 3.7 | Sample problem instance from MS-MACRO. | 20 |
| 3.8 | Example of a QA pair in RoITD. [149] | 23 |
| 3.9 | Question development user interface illustrating the annotation pro- cess. [149] | 24 |
| 3.10 | Following question formulation. | 24 |
| 3.11 | Question type distribution. | 27 |
| 3.12 | Statistics of Question Classification Dataset based on type of ques- tions. | 29 |
| 3.13 | (a) PCA 2D Visualization of BoW. (b) PCA 3D Visualization of BoW. | 32 |
| 3.14 | (a) PCA 2D Visualization of TF-IDF. (b) PCA 3D Visualization of TF-IDF. | 32 |
| 3.15 | (a) TSNE 2D Visualization of BoW. (b) TSNE 3D Visualization of BoW. | 32 |
| 3.16 | (a) TSNE 2D Visualization of TF-IDF. (b) TSNE 3D Visualization of TF-IDF. | 33 |
| 3.17 | Statistics of Question Classification Dataset based on type of ques- tions. | 38 |
| 3.18 | Outlier and inlier distribution of the MNIST dataset. [107] | 42 |
| 3.19 | Outlier detection performance of various models on CreditCard- Fraud raw dataset. [107] | 46 |

| | | |
|------|--|----|
| 3.20 | Zero-shot outlier detection performance of VAE-GAN-based method on CreditCard-Fraud dataset. [107] | 46 |
| 3.21 | F1-scores of VAE-GAN based method compared with other zero-shot outlier detection methods for the MNIST dataset. [107] | 47 |
| 3.22 | F1-scores of VAE-GAN based method compared with other zero-shot outlier detection methods for the KDDCUP99 dataset. [107] . . | 47 |
| 3.23 | VAE-GAN based method supervised outlier detection performance for the CreditCard-Fraud dataset. [107] | 47 |
| 3.24 | ROITD Alphagan Test Loss Distance. | 50 |
| 3.25 | Alphagan Vizualization. | 51 |
| 4.1 | Unfolded recurrent Neural Network [124]. | 54 |
| 4.2 | LSTM cell structure with an internal gates (Taken from [72]). | 55 |
| 4.3 | An overview of different categories of transfer learning (Taken from [154]) | 59 |
| 4.4 | Representation of self-attention mechanism in a sentence. | 61 |
| 4.5 | Transformer Architecture [209]. | 62 |
| 4.6 | Illustration of multi-headed self-attention [209]. | 63 |
| 4.7 | Input Illustration for BERT [64] | 65 |
| 4.8 | Downstream tasks fine-tuning using BERT [64] | 66 |
| 4.9 | Increasing hidden size of BERT- large leads to worse performance on RACE [1] | 67 |
| 4.10 | Increasing hidden size of BERT- large leads to worse performance on RACE [122] | 67 |
| 4.11 | The configurations of the main BERT and Albert analyzed [122] . . | 68 |
| 4.12 | Pipeline (top) compared with end-to-end (bottom) ASR models. For pipeline ASR, feature extraction is required, and different output representations are produced during decoding. For end-to-end ASR, raw waveforms are directly transformed into text. Rescoring can be completed in both cases using an added LM. [81] | 71 |
| 4.13 | Various speech features obtained for each processing stage. [81] . . | 73 |

| | | |
|------|---|----|
| 4.14 | Comparison of popular speech data sets for ASR applications. This table compares speech type and data set size in terms of speech hours and speaker number. Availability is shown for popular ASR frameworks, Kaldi (K), PaddlePaddle DeepSpeech (P), Wav2Letter (W), RWTH Returnn (R), and Nvidia (N). [50, 81, 84, 155, 157, 176] | 80 |
| 4.15 | Architectures and feature types of Kaldi-based ASR systems. Feature type 1 (upper left) is implemented within TDNN models (center) while type 2 features (middle left) are implemented within CNN-TDNN models (right). Output blocks (bottom left) using cross-entropy functions along with chain loss functions are utilized in both network types. [81] | 81 |
| 4.16 | RWTH RETURNN model (left) and PaddlePaddle DeepSpeech2 model (right). [81] | 82 |
| 4.17 | Facebook Wav2Letter networks comprised of completely convolutional architecture along with an ASG loss function (left) and encoder-decoder with TDS blocks (right). [81] | 83 |
| 4.18 | Nvidia time convolutional networks: Jasper, which contains dense residual blocks (left), and QuartzNet, which contains time-channel separable residual blocks (right). [81] | 84 |
| 4.19 | Comparison of ASR Systems. Systems are compared by (i) type—hybrid, HMM-based, and end-to-end type neural network, (ii) component types—multi-component and single neural networks containing additional or optional LM, (iii) speech features, (iv) neural network architecture, which also includes function loss, (v) output type and size, and (vi) model complexity regarding size of the model, quantity of activations, and operation count required to process a single speech frame. Systems are described with respect to complexity, architecture, and hardware requirements. [81] | 85 |
| 4.20 | Formulas used to determine network complexity. [81] | 86 |
| 4.21 | Comparison of ASR systems regarding performance where performance is measured by rate of word error. This evaluation is performed using two LibriSpeech subsets—test-clean and test-other. For frameworks that allow, two scenarios are used for each evaluation—with and without an external LM. [3, 4, 8–11, 13, 14, 81] | 86 |

| | | |
|------|---|-----|
| 4.22 | Librispeech corpus—various training and evaluation subsets, along with their corresponding size. [81] | 87 |
| 4.23 | ASR performance compared with hardware requirement trade-off. Performance is shown as the WER obtained using LibriSpeech test-clean dataset. Hardware requirements are shown with respect to memory load (MB) and minimum throughput (GOPS). Note: regarding the memory load, only the required amount to load the neural mode and store all activations for 1 second of speech processing are considered. More memory may be required for other components. Regarding throughput, only the operations needed to pass speech through the network were considered. | 88 |
| 4.24 | ASR performance compromises between memory requirements (left) and throughput requirements (right). [81] | 88 |
| 5.1 | Select Cloud run from Google Console. | 103 |
| 5.2 | Create Service in Cloud Run. | 104 |
| 5.3 | Connect Github to Cloud Run. | 104 |
| 5.4 | Flask App for calling the html web page. | 105 |
| 5.5 | Demo of Romanian BERT QA model in webpage. | 105 |
| 5.6 | Integration of webhook into the fulfillment of Dialogflow. | 107 |
| 5.7 | DialogflowSourceCode code. | 108 |
| 5.8 | SimpleSayReaction code snippet. | 109 |
| 5.9 | EmptyChatbotReaction code snippet. | 110 |
| 5.10 | DialogflowChatbot class. | 110 |
| 5.11 | onRobotFocusGained code snippet. | 111 |
| 5.12 | QA demo in Pepper. | 111 |
| 5.13 | Architecture of Haystack [5] | 112 |
| 5.14 | Semantic Search System [6] | 113 |
| 5.15 | Architecture QA model using Haystack. | 114 |
| 5.16 | Comparison of EM and execution time in seconds for selected models. | 122 |
| 5.17 | Comparison of EM and execution time in seconds for selected models. | 122 |
| 5.18 | Joint reasoning between the language model and knowledge graph (green box) for a given question-answer (purple box) [219] | 123 |

| | | |
|------|--|-----|
| 5.19 | Overview of the QA-GNN model where a provided QA (z) is connected with a sourced KG to create a joint graph, also referred to as a working graph. The relevance for each node in the KG is conditioned on z, and reasoning is performed on the working graph. [219] | 124 |
| 5.20 | Algorithm details of knowledge graph embedding-based domain adaptation. [70] | 125 |
| 5.21 | Overview of the algorithm. [70] | 128 |
| 6.1 | Token scores with vs without bayesian update. | 137 |
| 7.1 | Entailment tree formation process. The given hypothesis (green), which summarizes the QA pair, is combined with a corpus or relevant text to create an entailment tree. The entailment tree includes intermediate nodes (blue) to illustrate how the hypothesis is connected to the corpus. [57] | 143 |
| 7.2 | Two medium-complexity entailment tree examples. The hypotheses are the root nodes, identified as H (green). Intermediate conclusions are shown in blue. [57] | 144 |
| 7.3 | Sample questions with variable answers that depend on the geographical or temporal context. [231]. | 145 |
| 7.4 | Summary of seven natural logic relations. [136, 188] | 146 |
| 7.5 | Example of the natural logic proof process, which begins with the provided hypothesis "rodents consume plants" to determine the premise "squirrels eat nuts". Labels located at the node edges define the logical relationships between the associated sentences. [188] | 146 |

Acknowledgements

First and foremost I am extremely grateful to my supervisor, Acad. Dan Tufis for his invaluable advice, continuous support, and patience during my Ph.D. study. Their immense knowledge and plentiful experience have encouraged me all the time in my academic research and daily life. I would also like to thank the members of the Guidance Board Dr. Radu Ion, Dr. Horia Cucu, and Dr. Tiberiu Boros for their technical support in my study. It is their kind help and support that has made my study a wonderful time. Finally, I would like to express my gratitude to my wife, Laura. Without her tremendous understanding and encouragement in the past few years, it would be impossible for me to complete my study.

Chapter 1: INTRODUCTION

Artificial intelligence has made significant advances in natural language processing in recent years (NLP). The majority of development corporations are particularly interested in NLP applications for QA tasks. QA is the process of a user submitting a natural language-formatted query, which the language model subsequently answers concisely and correctly. A perfect QA model would be able to interpret questions in real time with minimal training data. As a result, text interpretation is required to obtain a relevant response to the question. The process of creating a suitable response entails sorting through a huge number of brief text fragments and categorizing them according to their calculated importance for the question. As a result, the task is to collect these text fragments and categorise them in order to answer the question while keeping the conversation's context in mind.

Hence in this project, we aim to develop a QA systems that is not only able to search the correct documents but also provides accurate answers to natural language questions. In order to answer the questions in natural language, there is the necessity of more complex text processing than that of currently available in information retrieval system. The background and user knowledge play a part in determining the answer because there are various ways to describe the same answer. This makes it difficult for the QA system to identify whether the two answers are equivalent (by combining and sorting responses), as well as to assess the QA system's results. Finding answers to definition inquiries is challenging because the term being defined is the only phrase available to focus the search. These terms are frequently used in papers without being defined first. The application of a BERT model trained using a big question-answer pair in the Romanian language is discussed in this article. Each question is self-contained and will be answered independently of the previous one.

For designing a QA system, we will look at various QA, information retrieval, and MRC datasets that are currently available. We'll look at the differences in datasets in terms of question domain breadth. We discuss various benchmark datasets such SQuAD, QUASAR, TriviaQA, MS-MACRO, RACE, Narattive QA, and MCScript. We also explore Open-Domain QA datasets such as DrQA and Wikipedia QA. In addition to this, we use an specific QA dataset designed on Romanian language called Romanian QA Task. A Romanian IT Dataset (RoITD) is a natural language dataset in Romanian with 9575 QA pairs created by crowd workers. Apart from the basic RoITD dataset, we extend this to classification task as pre-information retrieval technique. We use the RoITD dataset to build a domain classification depending on the type of query. Question classification is

an example of such a task. It assists in restricting the search area to a certain sort of query, allowing for much better response in QA tasks. We manually labelled this Romanian Question Classification (RQC) datasets based on the type of questions. The types used are “Ce”, “Cum”, “De ce”, “Care”, “Unde”, “Când”, and “Caror”. For making classification efficient we remove these WH question for training the model.

We then evaluate the RQC task on various classifier such as Support Vector Machine, Naive Bayes, Decision Trees, and Tsetlin Machine. We basically select these baselines because of their simplicity and interpretable mechanism. Since this RQC is a simple classification task, and the motivation is to design a simple and interpretable model. Here we have used Macro-F1 and Micro-F1 because of unbalanced labels in RQC. From the experiments we show that all the classifier performs quite similar except Naive Bayes which performs poorer than the rest of the model. Result suggests that Naive Bayes are learning only the samples with highest number of labels ignoring the smaller class because of low Macro-F1 and Micro-F1. Even though the performance of other models are similar, Tsetlin Machine outperforms all of them with slight margin in both Macro-F1 and Micro-F1.

In addition to this we discuss various existing outlier Detection methods mostly focusing on VAE-GAN Based Zero-shot Outlier Detection. This study developed a unique generative deep learning model based on Generative Adversarial Networks (GAN) and Variational AutoEncoders (VAE) that uses uniform distributions generated by variational autoencoders to distinguish inliers from outliers. We then discuss the results on RoITD outlier detection. We demonstrate the performance of different outlier detection algorithms on RoITD dataset along with the Alphagan Test Loss Distance. At last we use Alphagan Visualization to demonstrate the data points and Outliner scores.

We then propose different pipelines to extract the answers. We propose to use Haystack as the platform for huge document collection to design an end-to-end QA system. One of the main criteria to use Haystack is the ability to integrating the pretrained models from Huggingface. We used RoITD dataset to train various QA model that includes different variants of BERT and their variants. In order to design an chatbot interface, we implemented model in Google Cloud Services (GCS). GCS is then used in conjunction with Google Dialogflow to create a chatbot interface. Pepper is then linked to the dialogflow, which collects the response and displays it to the user. We use Bayesian Inference to improve the QA system. Bayesian inference is used to find the next best response. We were able to confirm our hypothesis that suppressing the top prediction and up-dating the likelihood of the remaining tokens would cause the model to explore more choices, resulting in a higher recall.

Finally, we looked at knowledge graphs for answering questions. Despite substantial improvements in QA processes over the previous decade, the most accurate contemporary methods combine the language model (LM) with the knowledge graph (KG). To combine these strategies, the model must be capable of extracting data from a large KG and conducting joint reasoning between the QA context and the KG structure. In all brevity, this thesis deals with extraction of data to cleaning it and training the QA model which eventually be used in real-time scenario as chatbot with the help of Bayesian Inference.

Here are the outlines of the contribution of the thesis:

- Development of the first Romanian QA dataset. In general datasets are extremely expensive to develop, and good quality ones require a lot of effort. Our dataset fill a need in the market and the fact that is open source opens the door for future research to be done. The dataset is studied in detail using multiple ML techniques as detailed in Chapter 3.
- Development of the RoAlbert Model. Training models for specific languages are time consuming and expensive and the fact that this model is open source enriches the Romanian NLP landscape. We have explained the model RoAlbert in the proposed in the implementation section detailed in Chapter 5.
- Integrating pepper with an real time open source QA model opens the door for future commercial applications out of the box, that can be deployed with ease in real world environment. We demonstrate this in Chapter 5.
- Bayesian modeling for QA is an relatively less studied domain, and the research presented in this document shows the potential of this type of approach in NLP. We are studying a very particular situation, where the first answer is not correct, the feedback is received from the user and the model uses this newly received information to do inference on the spot without using expensive hardware. This is explained in Chapter 6.
- An extensive review of the STT approaches is made, that can be used as a starting point for future real world applications, In this context a Robot (in our case Pepper) can communicate with the user leveraging language. This a more natural form of communication that offers multiple advantages and also challenges. The real time application of it is shown in Section 5.2.3 of Chapter 5.

Chapter 2: MOTIVATION

2.1 Problem Definition

Recent developments in artificial intelligence has made large impact in natural language processing (NLP). Most major companies are especially interested in NLP applications for question answering (QA) tasks. Although QA has been the subject of scientific interest for over sixty years, recent NLP developments have led to important advancement in this area [25, 89]. QA involves the process of a user submitting a natural language-formatted question which is then concisely and correctly answered by the language model (LM). When dealing with user queries, NLP tools like chatbots should be capable of detecting intent. For example, when we pose the question “What is the weather in Bucharest tomorrow?”, a chatbot should be able to understand Bucharest as a location, tomorrow as a date and the user intent to know the weather. Unlike a standard search engine, QA systems must be able to provide a clear answer to the posed question as opposed to a list of relevant information that may or may not contain the question’s answer.

QA has become commonplace in many applications such as standard QA, dialog systems (i.e., chatbots), community QA systems, multimedia QA, and reading comprehension QA (RC-QA) [25]. In standard QA tasks, the answer is obtained from a knowledge base (KB) or free text. Dialog systems are designed to chat with an agent. Community QA systems allow users to post a question that can be receive several answers from the other users within the community. The community QA system is used to validate the suggested answers, then select the most accurate and/or relevant one. Multimedia QA poses questions in the form of an image or video, meaning that the system must be capable of multimedia processing and rational. For RC-QA systems, the system is provided both a question and a passage where the RC-QA must be able to find the answer within the provided passage.

Reading comprehension (RC) tests the level of text understanding using questions and answers. Applying RC tests to natural language comprehension for computers is known as machine comprehension. During machines comprehension testing, machines are provided with a passage, then tasked with answering a question or question set. Questions can be provided in either multiple-choice or short-answer formats. The first RC-QA system is QUALM, which was developed by Lehnert in 1977 [25]. RC-QA can be especially challenging since it combines many difficult tasks like reading, comprehending, processing, reasoning, and answering. Some of the primary challenges that are still being studied include synthesis, paraphrasing, and inference [25]. Synthesis involves the integration

of information that is distributed over multiple sentences within a passage. Paraphrasing recognition requires word knowledge and synonymy to recognize sentences in a provided passage that include or paraphrase the provided question. Inference refers to the ability to provide an answer even when the passage information is incomplete. As shown in Figure 2.1, RC-QA systems are generally comprised of an embedding layer, encoding layer, attention layer, and output layer [25]. The embedding layer contains the input (i.e., question and passage) representation model, which is usually GloVe or Word2Vec. The neural network model uses the encoding layer to separately encode the question and passage, most often using a recurrent neural network (RNN) technique. The attention layer contains an attention mechanism that is used to capture relations between the passage and question. Finally, the output layer is where the answer is generated or found.

The three primary QA system paradigms are—(1) free text QA, also known as the information retrieval approach [73, 200], (2) the KB-QA approach [96, 224, 233], and (3) the hybrid paradigm [59]. Free text QA relies on question analysis to assess the type of answer required, followed by the implementation of Information Retrieval (IR) to identify the correct answer from a corpus. In KB-QA, questions are processed into predicates with semantic representations. Once this is complete, the system searches through a knowledge base (KB). Hybrid paradigm approaches use both free text and a KB to provide wider system coverage to improve the probability of producing a correct answer. These systems are evaluated using various metrics such as recall and precision that are generally dependent on the track or application. Generally, QA systems categorize questions as either factoid or non-factoid [25]. Factoid questions are highly specific and have a single answer. Non-factoid questions, also referred to as complex questions, are open-ended with many correct answers which requires the QA system to be capable of reasoning. DL methods have been successfully developed for RC-QA models.

The two primary KB-QA types are Multi-Relation questions and Single-Relation questions [25]. Multi-Relation questions determine the system’s ability to answer constrained questions. Six primary constraints have been recognized in the literature [30]. Three large datasets are available for KB-QA use, all of which are based on Freebase KB, meaning that each of questions are answerable by Freebase [25]. These are the SimpleQuestions dataset [100], which is directed towards single-relation questions, the WebQuestions dataset [34] used for multi-relations questions, and the ComplexQuestions dataset [30] used for multi-relations questions. Webquestions is designed to answer real questions. A summary of different KB-QA directions is provided in Table 2.2.

KB-QA systems still face many challenges when it comes to answering complex ques-

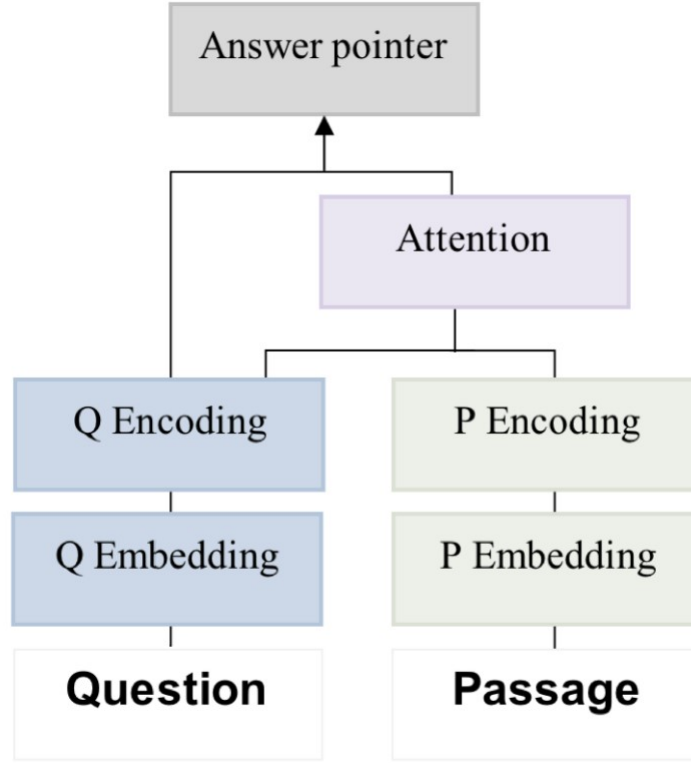


Figure 2.1: Sample DL RC-QA model architecture. [25]

| Dataset | State of the art | Evaluation metrics | Methodology | Limitations |
|---------------|---------------------|--------------------|---|----------------------------|
| WebQuestions | (Yih et al., 2015) | F1=52.5% | SP-based..CNN for semantic similarity | Handcrafted features |
| WebQuestions | (Hao et al., 2017) | F1=42.9% | IR-based.. Bi-LSTM with cross attention model | |
| TempQuestions | (Jia et al., 2018b) | F1=36.7% | SP-based.. | Hand-coded query templates |

Figure 2.2: Summary of recently reported KB-QA methods. [25]

tions, especially during real-time customer service interactions when users provide specific information and require immediate answers. These scenarios often include three types of questions—(1) questions involving specific conditions, (2) intention-based questions, and (3) questions that require constraint interference. Although KBs are extremely large, they are not completely comprehensive and will be missing many facts. Additionally, lexical and vocabulary gaps often exist in KBs where the vocabulary of the user does not match the vocabulary of the KB [25]. Knowledge completion techniques for QA systems have been developed as a way of bridging any knowledge gaps. DL has been a powerful way to overcome these issues. Word embedding, for instance, is a method to bridge incomplete and vocabulary gaps [25]. Figure 2.3 summarizes the available benchmark datasets that include complex questions and introduces the three relevant general domain KBs.

WebQuestions [34] is designed to answer real questions. The database questions are

collected from the Google Suggest API. Answers are annotated using Amazon Mechanic Turk. Although this is the most popular dataset, it has two major disadvantages [76]. The first is that only answers are labeled for questions with no provided logical forms. The second is that 84% of the database is formed of simple questions with only a few of those requiring constraint inference and multi-hop reasoning. WebQuestionsSP [222] is designed to solve the first issue by annotating SPARQL query statements in WebQuestions and removing questions with unclear intentions, ambiguous expressions, or unclear answers [76]. ComplexQuestions [30] targets the second issue by introducing additional questions containing type or entity constraints, implicit or explicit time constraints, and aggregation constraints. Finally, ComplexWebQuestions [199] is used to modify the WebQuestionsSP SPARQL queries by adding constraints, then generating natural language questions using Amazon Mechanic Turk and templates.

Question Answering over Linked Data (QALD) is an evaluation subtask for the Conference and Labs of the Evaluation Forum (CLEF) [76]. Since its initiation in 2011, it has been held annually and provides many training datasets in addition to testing these sets each time. Complex questions account for approximately 38% of the questions including those with multiple entities and relationships.

Large-Scale Complex Question Answering Dataset (LC-QuAD) [202] was published in 2017 based on DBpedia. Approximately 82% of the dataset is composed of complex questions which are constructed using predefined SPARQL templates. These templates are initially filled with seed entities and associated relations. These are used to generate the specific SPARQL queries on DBpedia. The queries are then converted to natural language questions using a combination of crowdsourcing and predefined question templates. By expanding on this framework, a more diverse and larger KBQA dataset was formed, named LcQuAD 2.0 [69], containing more forms of complex questions based on DBpedia and Wikidata.

| Dataset | Background KB | Size | Logical forms |
|-------------------------|-------------------|--------|---------------|
| WebQuestions [1] | Freebase | 5,810 | No |
| ComplexQuestions [2] | Freebase | 2,100 | No |
| WebQuestionsSP [3] | Freebase | 4,737 | Yes |
| ComplexWebQuestions [4] | Freebase | 34,689 | Yes |
| QALD | DBpedia | 50-500 | Yes |
| LC-QuAD [5] | DBpedia | 5,000 | Yes |
| LC-QuAD 2.0 [6] | DBpedia, Wikidata | 30,000 | Yes |

Figure 2.3: Summary of KB-QA benchmark datasets that involve complex questions. [76]

Name-entity recognition (NER) plays an important role in extracting such information

from sentences. Chatbots use intentions and entities to understand user-written text. For humans this is a natural process, but for machines this is a complicated task that requires large amounts of data and computing power to achieve marginal results. The primary problem lies in real-time text analysis, especially when there are a limited amount of training examples. Despite the fact that deep learning-based techniques have achieved state-of-the-art results for several NLP tasks, these methods still suffer from insufficiently labeled training data.

DL architecture has been studied to improve NER and intent classification accuracy. New performance records are created for various NLP tasks every few months. Higher accuracy means answering customers correctly most of the time. BERT (Bidirectional Encoder Representations from Transformers) is a large neural network model pre-trained using over 3 billion English words sourced from Wikipedia. The extensive pre-training of BERT also offers support of 104 different languages. The BERT model uses context from both sides adjacent to each word (i.e., the left and right hand sides) to determine or predict its meaning. For example, the meaning of a word like "bank" is different when using the terms "river bank" and "bank account". BERT has already shown the best results, beating state-of-the-art in eleven NLP tasks. We are attempting to use the BERT model for intent classification. Since BERT has been so extensively pre-trained, we are going to explore the pre-trained Romania BERT model on an IT-focused dataset prepared in the Romanian language.

To solve the previously mentioned problem, we need a model capable of interpreting questions in real-time using less training data. Therefore, text interpretation is a must to get a meaningful answer from the question. Specifically, the chatbot should be able to perform an intent classification to get the meaning of a sentence whenever a user says something. As with human dialogue, chatbots usually speak for multiple rounds. The purpose of the task is to respond correctly to your request, as the dialog becomes the context. In this case, generating the appropriate answer involves a large set of short text fragments and classifying them according to the calculated importance for the query. Therefore, the task is to fetch these text fragments and classify them to answer the question by taking the context of the conversation into account.

The major aim in QA systems is to develop a technology that not only searches the correct documents but also obtains accurate answers to natural language questions. To answer questions in the natural language, more complex text processing is required than what is currently employed in the information retrieval system. Some response systems that can be handled have been developed. This is required to achieve improved accuracy.

However, there are few reports of technologies that can quickly find the correct answer.

2.2 Question Answering Procedure

If we think of QA as a human activity, what do we expect when someone is asked a question to which they do not know the answer? In this case, the respondent may consult some other knowledge source (e.g. books, libraries, internet, etc.) to find information they can read and understand to determine the answer. Then they can return to the person who originally asked the question and tell them the answer. They can also indicate where to find the answer so that the questioner can trust the relayed information. What they will not do is just return a book or a several documents (which may hold the answer) as a question response. Presumably, this is what most computer users must do now in NLP.

Many people hope to use the internet as a source of knowledge from which they can find answers to their questions. Although many search engines, for some cases, suggest that you can ask questions, the returned results are usually part of several documents, which may or may not contain answers, that have a lot in common with the question. This is because the complex text processing and understanding requirements placed on QA models exceed the capabilities of widely available search engines.

Three primary approaches are used in KB-QA—semantic parsing (SP) or neural semantic parsing (NSP), information retrieval (IR), and deep learning (DL) [25, 96]. SP-based methods rely on semantic parsers to reconstruct questions into structured expressions (e.g., logical forms) [221]. SP is also used to create structured queries from natural language questions (SPARQL). SPs form executable query languages from natural language questions [76]. NSPs, meanwhile, use neural networks to construct SPs in order to improve the parsing scalability and capability. Unstructured questions are mapped to structured logical forms. These can then be transformed to executable queries using manually-crafted rules. NSP-based models generally slightly outperform most IR-based methods [76]. IR-based methods, also known as relation extraction-based methods, uses the input question to identify answers from a KB. These method rely on binary classification or candidate sorting for QA [76]. First, candidate answer and question distributed representations are generated. The matching score between these encoded questions and answers are then calculated to choose the final answer. Some IR methods use a multi-hop reasoning framework to deal with complex questions. This type of methodology remove manually defined templates and rules, however they are unable to interpret the model. Additionally, they are unable to process complex questions that require constraint inference. Finally, DL-based methods,

also known as embedding-based methods, represent questions and answers using semantic vectors [96]. The most similar answer is then identified by applying a similarity matrix.

Recent methods rely on three primary stages—topic entity, fact finding, and answer selection [25]. During topic entity, the topic of the question is identified. This can be completed using an API [96, 221]. Fact finding, also referred to as relation extraction, searches for relationships between the defined topic entity to provide candidate knowledge triples. This can be completed using knowledge completion [221]. Finally, answer selection is used to match the identified question and candidate triples to form semantic vectors. The semantic relevance score is then calculated between these two fields using a predefined similarity measure to choose the most similar answer [96].

Traditional KBQA methods generally use predefined templates or rules to parse questions in order to obtain logical forms. This includes the standard bottom-up parser [34]. In this method, KB entities or relations are coarse mapped from question phrases using a large text corpus and KB [76]. When provided a question, the parser recursively calculates using lexicon KB relationship and entity mapping to question phrases along with four hand-crafted operations (i.e., Interaction, Join, Aggregate, and Bridging). The parser uses a log-linear model opposed to the manually-defined features to separate from poor calculations while minimizing the search space.

A template-based model, Aquu, has been developed for question mapping to three templates (Fig. 2.4 [76]). First, any entities from the KB matching a portion of the question are identified. Matches can be either literal or an alias of the entity. Aquu then uses the three templates where the KB subgraph is centered on the matches. The ranking model based on manually defined features is then used to select and output the best matches to query the KB and obtain a question answer.

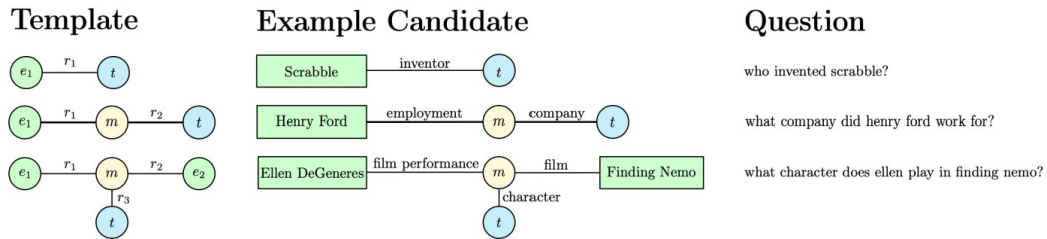


Figure 2.4: Question mapping templates in Aquu including example candidates and corresponding questions. [31, 76]

Temporal language is comprised of time, event, and temporal relation that can be defined by three primary categories—temporal relation (TLINK), subordinate (SLINK), and

aspectual (ALINK) [25]. TLINK is representative of the temporal relationship between an event and a time, two events, or two times. SLINK is used to represent evidential and factual modality. ALINK only describes the relationship between two events, or aspectual connections. The relational extraction portion of temporal representation is the most difficult. Temporal QA is used to answer temporal-based questions, which involves the combination of temporal information extraction and reasoning. Temporal questions can be categorized as temporal answer, explicit temporal, implicit temporal, and ordinal constraints. Temporal answers involve questions that inquire about time or date. Explicit temporals involve questions that include explicit date, time, or event information. Implicit temporals are where the question does not have an explicit temporal term, but instead contains the terms before, during, or after. Ordinal constraints require a rank to answer a question.

2.3 Challenges of the Question Answering system

What makes question-answering so difficult in NLP? The major question remaining is “what has to be programmed in a computer to make it capable of answering questions as humans do”? Even if it is assumed that all QA system input is provided in a question format, wider dialogue remains a problem, because there are usually many ways to ask the same question. Sometimes the user’s turn is simply a statement of the problem, and sometimes this statement depends on the environment in which the question is raised or the knowledge of the questioner. Similarly, there are multiple ways to describe the same answer, so the background and user knowledge plays a role in defining the answer. For example, the response “last Wednesday” is meaningless without knowing the current date or the date the response source was recorded. This is not only difficult for the QA system to determine whether the two answers are equal (combining and ranking responses) but is also challenging to evaluate the QA system results. The versatility of natural language extends to both question and answer expressions, making it difficult to ensure the training example containing the answers to the given questions is retrieved.

When asking definition questions (e.g. “What is machine learning?”) with little change, QA system answers are usually more diverse than those of a factual question (e.g. a person’s name or date). Finding answers to definition questions is very difficult since the only available phrase to narrow the search is the term being defined. These words are often used in documents without initially defining them. This means that many training samples referring to the defined terms will be useless to the QA system, because it is trying to create a definition. Obviously, the development of computer systems capable of answering ques-

tions poses many challenges. In this thesis, we discuss the application of a BERT model trained using a large question-answer pair in the Romanian language. Each question is independent and will be answered completed separately from the previous one. Of course, this removes some of the issues discussed above, especially that of context, allowing us to freely focus on other issues. Furthermore we propose and implement end-to-end QA system and we deploy it on the robot Pepper. Lastly, we investigate leveraging Bayesian methodologies is the QA setup, with minimal computational overhead, focusing on the situation when the first answer given by the system is not correct. We experiment with different ways of using the new information given (that the initial answer is not correct).

Chapter 3: DATASET

In this chapter, we will cover the basic concepts of question answering datasets, types of datasets in QA model, reading comprehension QA, and newly introduced Romanian datasets with its variant.

3.1 Introduction

The initial method of answering questions in natural language involves analyzing the query grammatically and then extracting the response using a set of pre-defined rules. This strategy was based on the successful application of rule-based transformation of questions in ELIZA [216]. ELIZA was designed to turn questions back on the person asking them—for example, “Who are you?” the program might respond, “Why are you interested in whether I am or not?” This necessitates a grammatical analysis of the query, which following programs like Baseball [88], one of the first question-answering systems, built upon.

Rule-based systems were popular until the late 1990s, and some are still being developed today. Designing a rule-based system, on the other hand, necessitates enumerating every potential question variant or defining a formal grammar to which all questions must adhere. In the same way, information extracted from natural language text must be in a specified format. These systems are incapable of dealing with the complexity of natural language and cannot be said to understand text. Nonetheless, to enhance the input features, analytical approaches designed for rule-based systems, such as part-of-speech taggers or named entity recognizers, are still in use [45]. Rule-based systems frequently impose strict limitations on the format of the question, but question answering research in general aims to eliminate as many of these artificial constraints as possible, allowing users to ask inquiries in totally natural language. While most datasets contain such challenges, some use cloze-style questions, a type of question that is halfway between artificial and freeform: A cloze-style question is one in which one or more important words have been blanked out. The word or phrase that correctly fills in the space is the "answer." Cloze tests have long been used in education to assess understanding in foreign language learning. Because the sentences that form the basis for the cloze are freeform, it acts as a bridge between a fixed format question and a freeform question. However, a cloze-style sentence provides more contextual clues to the answer than a question, as well as being more similar to information the system might find in context documents.

Language modeling has led to the critical development of advanced NLP tasks such

as machine reading comprehension (MRC). MRC requires the computational comprehension of natural language text to answer provided questions. This task is especially crucial for advanced search engines and intelligent agent frameworks at the cost of expensive datasets that are difficult to gather and often limited to a single language, most often English. Examples of the types of datasets used for English Question Answering models include SQuAD1.1. [167], SQuAD2.0 [166], and CoQA [169]. Although less common, other datasets have been developed in alternative languages such as Chinese (i.e., span-extraction MRC [54] and user-query-log-based DuReader [98]), Korean (MRC dataset [129]), and French (Question Answer dataset [67]) [149].

3.2 Various QA datasets

Here we will explore various available QA, information retrieval and MRC datasets. We will analyse the difference in datasets with respect to broadness of question domains. The grammatical variety of the questions, the extent to which the required information is distributed (whether the answer requires reasoning over multiple pieces of evidence), and whether answers are expected as free text, as a substring of the context, multiple-choice, or cloze-style fill-in-the-blank are also distinguishing features.

3.2.1 Stanford Question Answering Dataset (SQuAD)

One of the most widely studied question answering datasets is the Stanford Question Answering Dataset (SQuAD) [167]. On the exact match metric, the best submissions outperformed humans. As a result, it may be argued that the problem has been "solved," and that additional investigation of this dataset is unlikely to provide more intriguing results. However, because there are so many high-performing question answering systems on this job, it remains of impactful interest to this field.

In order to create the dataset, the authors selected 536 articles from the English Wikipedia from the top 10 000 articles by PageRank [153]. After removing tables, pictures, and paragraphs with less than 500 characters, there are 23 215 paragraphs left. Human annotators asked up to five questions for each paragraph using Amazon Mechanical Turk crowdsourcing, totaling 107 785 questions. SQuAD consists of contexts, questions, and answers as shown in Fig 3.1. In addition to the question, a paragraph of context is provided, and the solution is a substring from the text. The authors went one step further and collected up to four alternative answers for each question, all of which were correct. The paragraphs may have similar topics (since they could be from the same article), yet they cover a wide range

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

Question: Which NFL team represented the AFC at Super Bowl 50?

Answer: Denver Broncos

Figure 3.1: A sample from SQuAD.

of topics in total. SQuAD is not, however, an open-domain question-answering dataset. The purpose of open-domain question answering, according to the authors, is "to answer a question from a huge collection of documents," yet this dataset requires answer extraction, i.e. identifying the proper answer span in a single document.

Version 2.0 of the dataset was released in June 2018 [166], which adds a new class of question-context pair to the previous dataset: a problem instance in which the question cannot be answered using the paragraph provided, frequently due to a small semantic change.

3.2.2 QUASAR

QuesTion Answering by Search and Reading (QUASAR) [66] presents two datasets with two tasks: QUASAR-S and QUASAR-T. QUASAR-S (S for Stackoverflow) is a 37 000-statement cloze-style closed-domain question-answering task. The tasks were generated automatically using stackoverflow.com's programming help section. Each statement is the site's definition of a programming-related question tag (for example, java or.net). A placeholder is used in place of the tag. There are two versions of the dataset, each with a different type of context: One version contains 50 long parts (the full answer plus the full text of the question) and the other contains 200 short parts (one sentence) from the 50 most popular question threads related to the tag. Figure 3.2 shows a problem instance from

Question: javascript—javascript not to be confused with java is a dynamic weaklytyped language used for XXXXX as well as server-side scripting.
Answer: client-side
Context excerpt:
JavaScript is not weakly typed, it is strong typed.
JavaScript is a Client Side Scripting Language.
JavaScript was the ****original**** client-side web scripting language.

Figure 3.2: Sample problem instance from QUASAR-S.

Question: 7-Eleven stores were temporarily converted into Kwik E-marts to promote the release of what movie?
Answer: the simpsons movie
Context excerpt:
In July 2007, 7-Eleven redesigned some stores to look like Kwik-E-Marts in select cities to promote The Simpsons Movie.
Tie-in promotions were made with several companies including 7-Eleven, which transformed selected store into Kwik-E-Marts.
“7-Eleven Becomes Kwik-E-Mart for “Simpsons Movie” Promotion”.

Figure 3.3: Sample problem instance from QUASAR-T.

QUASAR-S.

QUASAR-T (T for trivia) is a database of about 54 000 human-written trivia questions covering a wide range of subjects. ClueWeb09, a collection of about one billion web pages, is used as a knowledge basis for answering these questions. The questions, on the other hand, were written and gathered independently from the corpus and came from various online sources. 50 lengthy (2048 characters) or 200 short (200 characters) context faux papers are chosen from the ClueWeb09 websites. It is not guaranteed that the answer string will be found in any context document due to the nature of context retrieval. Figure 3.3 shows a problem instance from QUASAR-T.

Although both QUASAR datasets provide more context than SQuAD, the papers are likely to be highly redundant, with the correct answer appearing in many context documents.

Question: The Dodecanese Campaign of WWII that was an attempt by the Allied forces to capture islands in the Aegean Sea was the inspiration for which acclaimed 1961 commando film?
Answer: The Guns of Navarone
Context excerpt:
Context excerpt: The Dodecanese Campaign of World War II was an attempt by Allied forces to capture the Italian-held Dodecanese islands in the Aegean Sea following the surrender of Italy in September 1943, and use them as bases against the German-controlled Balkans. The failed campaign, and in particular the Battle of Leros, inspired the 1957 novel The Guns of Navarone and the successful 1961 movie of the same name.

Figure 3.4: Sample problem instance from TriviaQA.

3.2.3 TriviaQA

TriviaQA by Joshi et al. [112] is similar to QUASAR-T in that it is made up of human-written trivia questions acquired from the internet on a variety of themes. There are 650 000 question-answer-context triplets in all, with 95 000 question-answer pairs annotated with on average six separate context documents. Despite the fact that the questions were produced in isolation from any context, the dataset includes two sets of context materials for each question. The first group is based on a crawl of the subject’s top Bing web search results, eliminating keywords like trivia, question, and answer. The second collection includes Wikipedia articles for any entities that were recognized automatically in the inquiry. Figure 3.4 shows a problem instance from TriviaQA.

The second collection includes Wikipedia articles for any entities that were recognized automatically in the inquiry. As a result, TriviaQA provides a lot of information about each question. The authors also give an unfiltered version with 110 495 question-answer pairings and 740 000 context documents (context documents that do not contain the response string have not been filtered away). In comparison to SQuAD, even the filtered version offers a far bigger background for each question. TriviaQA’s average context length is 2 895 words, making many techniques that would work on SQuAD unable to apply to this dataset.

3.2.4 MS-MARCO

The Microsoft MARCO (MACHINE Reading COMprehension) dataset contains 182 000 online queries and replies (in the most recent version at the time of writing). The ques-

| |
|---|
| <p>Query: will I qualify for osap if I'm new in canada</p> <p>Context (excerpts): you must be a: 1. Canadian citizen; 2. Permanent resident or 3. Protected person/convention refugee You will not be eligible for a Canada-Ontario Integrated Student Loan</p> <p>Answer: No. You will not qualify.</p> |
|---|

Figure 3.5: Sample problem instance from MS-MARCO.

tions are real queries submitted to the Bing search engine by users, which Microsoft has collected and anonymized. Crowdworkers were urged to write full-text answers, so they wrote them. They were given the top search results from the given query as context. These identical extracts are also included as context documents in the dataset. Each question is also annotated with the URLs of the web pages from which the context snippets were derived, as well as a categorization tag, such as ENTITY or NUMERIC, indicating the type of response expected. This tag is meant for model analysis rather than training. Figure 3.5 shows a problem instance from MS-MARCO.

3.2.5 RACE

The RACE dataset [121] contains 28 000 english passages and 100 000 questions created by English instructors for Chinese middle and high school English classrooms. There is a huge difference in difficulty across the questions because the examination questions are aimed at students aged 12 to 18. As a result, the authors advise splitting the dataset into RACE-M, which includes questions for middle schoolers (12-15 years old), and RACE-H, which includes high school questions (15-18 years). Each question has four multiple-choice options. As a result, the authors advise splitting the dataset into RACE-M, which includes questions for middle schoolers (12-15 years old), and RACE-H, which includes high school questions (15-18 years). Each question has four multiple-choice options. Figure 3.6 shows a problem instance from RACE.

3.2.6 MS MACRO

MS MACRO contains 1,010,916 questions, 8,841,823 companion passages taken from 3,563,535 web documents, and 182,669 editorially prepared responses. This is a large-

Context (excerpts):

Many people optimistically thought industry awards for better equipment would stimulate the production of quieter appliances. It was even suggested that noise from building sites could be alleviated

Question: What was the author’s attitude towards the industry awards for quieter? (sic)

Answer: A: suspicious, B: positive, C: enthusiastic, D: indifferent

Figure 3.6: Sample problem instance from RACE.

scale dataset from the real world. It consists of Bing or Cortana search queries that have been anonymised. They present a series of extracted paragraphs from documents collected by Bing in response to the question for each question. The passages and documents may or may not include all of the information required to solve the question. Crowd-sourced editors must generate answers for each question based on the information found in the recovered texts.

3.2.7 Narrative QA

This dataset contains questions based on summaries of movie screenplays and books that were prepared by editors. There are approximately 45,000 question-answer pairs distributed throughout 1,567 stories in the dataset, which is evenly split between books and movie scripts. This dataset includes tasks based on both summaries and full novels/scripts. NarrativeQA’s full version requires you to read and comprehend entire stories (i.e., books and movie scripts). This task is currently intractable for existing neural models out of the box, according to the authors. One of the datasets for which we develop models in this thesis is this one.

3.2.8 MCScript Dataset

According to this dataset given as part of the SemEval Task 2018, the introduction of commonsense information benefits natural language understanding systems. Authors concentrate on scripts, which are commonsense information about everyday actions. Scripts are events that describe stereotypical human behaviors (also known as situations), such as baking a cake or boarding the bus. Many times, context is missing or insufficient infor-

Text: My backyard was looking a little empty, so I decided I would plant something. I went out and bought tree seeds. I found a spot in my yard that looked like it would get enough sunshine. There, I dug a hole for the seeds. Once that was done, I took my watering can and watered the seeds.

QUESTION/ANSWER:: Why was the tree planted in that spot?

1. to get enough sunshine.
2. there was no other space.

While the above question should be easily answerable from text, the following question answer pair requires common sense knowledge, which can be incorporated into the model in various ways and is explored in further depth as part of this thesis in later sections.

QUESTION/ANSWER:: What was used to dig the hole?

1. a shovel.
2. their bare hands.

This is one of the few large scale datasets that help tackle the challenge of augmenting model’s reasoning capabilities with common sense knowledge and hence is a novel task.

Figure 3.7: Sample problem instance from MS-MACRO.

mation is available to resolve the ambiguity. In such instances, the authors believe that including commonsense knowledge about the world in an NLU system might be beneficial. There are 2,119 texts and 13,939 questions in total. The texts in the data set comprise 110 script scenarios of varying complexity and talk about everyday activities. Figure 3.7 shows the example of the dataset.

3.3 Datasets Comparison

For model development, ideal datasets should include the following characteristics:

- 1. Large enough for large deep learning models to be trained.
- 2. Contain questions about the underlying narrative rather than the text’s surface form, which should necessitate the construction of more abstract representations about the actions and relationships portrayed throughout the document. Answering such questions necessitates readers combining information from numerous statements throughout the document and generating a coherent response based on this combined information. That is, they assess the reader’s ability to grasp language

rather than just pattern matching.

- 3. Question-and-answer pairs should be more representational of a normal distribution of knowledge needs that users might desire to meet with the help of, say, an intelligent assistant.

SQUAD, MS MARCO, and Narrative QA are three datasets that need replies to be more than single words/entities and present a more difficult task than a simple answer ranking problem. SQUAD presents a huge number of questions from these three databases. However, these come from a tiny number of documents, each of which is only a few pages long, limiting the lexical and subject diversity that models trained on this data can handle. On both SQUAD and MS MARCO, simple models scoring and/or extracting candidate spans conditioned on the query and superficial signal from the rest of the document perform well. These models will not easily generalize to issues where the solutions are not document spans or where a correct answer requires numerous discontinuous spans. The span prediction models in NarrativeQA’s baselines for summary-based questions are also the best-performing. However, there is still a compelling disparity between the answers provided by humans and the projected spans.

Furthermore, in Narrative QA, fictional stories have a number of advantages as a domain because they are largely self-contained, summaries have more intricate linkages and timelines than news pieces or brief paragraphs from the web, and so provide a task that is different in nature. Real-world text, on the other hand, is messy: it may contain mistakes, abbreviations, and transcription problems in the case of spoken interfaces. SQUAD and NarrativeQA both have high-quality stories or text spans from Wikipedia and other sources. Machine reading comprehension systems in the real world should be tested on genuine datasets in which they must be robust to noisy and problematic inputs. MS MARCO solves this problem because the questions are based on actual search queries given to Bing by users. Furthermore, none of these datasets specifically evaluate a model’s commonsense reasoning abilities, which is critical when the necessary context for a query isn’t immediately provided in text and inferences from external knowledge bases are required. The MCScript Dataset overcomes this constraint.

As can be observed, each dataset has its own set of restrictions, and there is still need to produce more datasets to enable the modeling of real-world question-answering systems.

3.4 Open-Domain Question Answering

In open-domain question answering, the context is either not provided with the question or is provided but is either too broad or too much for the question answering system to employ directly. Such systems must employ an information retrieval (IR) component to condense the available context into a manageable amount that can be fed into the question answering system (which is usually built on neural networks). We must compare our performance to the models from this section because this task is the focal point of this project. However, exact numbers are often impossible to compare since various datasets are used or varying amounts of context are gathered.

3.4.1 DrQA

DrQA [45], which stands for Document Reader Question Answering, is a typical system for this problem: TF-IDF [181] is used to extract the top five articles from a corpus of Wikipedia articles. Those articles are divided into paragraphs and fed into the question answering system, which aggregates response candidates before selecting the most likely answer from all paragraphs from all articles.

3.4.2 Wikipedia QA

Ryu, Jang, and Kim [178] provide another methodology for answering questions utilizing Wikipedia as a knowledge source. It's a rule-based system without any machine learning algorithms. Instead, it uses pre-defined regular expressions to extract structured information from Wikipedia articles, utilizing the structure that is explicitly incorporated in Wikipedia. While they can answer over 85% of questions, the results are practically impossible to compare to other models because their model is hand-tailored to the Korean-language Wikipedia, and their test set consists of only 600 (unpublished) questions that they gathered themselves based on criteria that they do not report.

3.5 Romanian Dataset

3.5.1 Romanian QA Task

A Romanian IT Dataset (RoITD) comparable to SQuAD 1.1 has recently been introduced in the literature [149]. RoITD is a fully Romanian natural language dataset containing 9575 QA pairs designed by crowd workers [149]. The QA pairs were developed using 5043 IT- and household product-focused Romanian Wikipedia articles [149]. 5103 of the questions are 'possible', meaning the correct answer is within the provided paragraph while 4472 questions are 'not possible', meaning that the provided answer is only plausible and cannot be considered correct. Figure 3.8 shows a RoITD sample for reference. The dataset was evaluated by fine-tuning the QA model using the following transformer-based pre-trained models—(1) Multilingual BERT [65], (2) DistilBERT [183], (3) Romanian BERT (RoBERT) [138], and (4) XLM-R [53]. While transformer-based DL models can surpass human accuracy for MRC tasks, their performance in alternative languages has yet to be thoroughly tested.

```
1- {
2-   "version": "v1.0",
3-   "data": [
4-     {
5-       "title": "IT1",
6-       "paragraphs": [
7-         {
8-           "qas": [
9-             {
10-              "question": "Ce fel de telefon este iPhone SE",
11-              "id": "RM53H9975",
12-              "answers": [
13-                {
14-                  "text": "cel mai puternic telefon de 4 inchi din toate timpurile",
15-                  "answer_start": 25
16-                }
17-              ],
18-              "is_impossible": 0
19-            }
20-          ],
21-          "context": "Iti prezentam iPhone SE, cel mai puternic telefon de 4 inchi din toate timpurile. Pentru a-l crea, Apple a pornit de la un design de mare succes, apoi a reinventat tot restul. A9 este acelasi cip avansat folosit in iPhone 6s. Camera de 12 megapixeli face poze incredibile si filmeaza 4K, iar functia Live Photos iti aduce imaginile la viata. Rezultatul este un iPhone care pare mic, dar traieste pe picior mare. Un design mult iubit. iPhone SE preia un design incredibil de popular si il aduce la un nou nivel de rafinament."
22-        }
23-      ]
24-    }
25-  ]
26- }
```

Figure 3.8: Example of a QA pair in RoITD. [149]

Crowd workers created four maximum questions for a given paragraph where a minimum of one of these questions was unanswerable using an Amazon Web Service t2.micro machine with Ubuntu 20.04.1 operating system [149]. Inquiries avoided quoting the text and instead required original structure while still maintaining proper grammar [149]. The formulated queries were categorized using "U" (i.e., the question is not answerable with a likely answer) or "A" (i.e., the question is answerable with a correct answer). The user interface for question development is shown in Figure 4.4, and Figure 3.10 shows the con-

firmation following question registration.

Figure 3.9: Question development user interface illustrating the annotation process. [149]

Figure 3.10: Following question formulation.

RoITD follows a data structure format similar to SQuAD which includes "context", "id", "question", "text", "answer_start", and "is_impossible" attributes [149]. Paragraphs shown to crowd workers are stored within the "context" field. The "id" attribute consists of randomly assigned identification numbers unique to each question-answer pair. The "question" field contains the approved questions formulated by the crowd workers. The "answer_start" attribute tracks the character index that marks the answer beginning. The "is_impossible" attribute is defined using "0" or "1" where "0" indicates that the question is answerable with a correct answer (category "A") and "1" indicates that the question is non answerable with a likely answer (category "U").

Table 3.1: Categorization of 50 randomly chosen Question Answer pairs based on reasoning. [149]

| Reasoning | Description | Example | Percentage |
|------------------------------------|---|--|------------|
| Lexical variation | Major correlation between the question and the answer sentence are synonyms | Q:Ce se poate detecta cu ajutorul camerelor duale AI? Sentence: Cu camerele duale AI din spate de 13 MP + 2 MP, se pot recunoaste cu usurinta obiecte ... | 54% |
| Lexical variation(world knowledge) | Major correlation between the question and answer sentence requires world knowledge to resolve. | Q:Cu ce este asemantor Evelatus? Sentence: ...un telefon similar cu Evelatus Easy ... | 12% |
| Multiple sentence reasoning | There is anaphora, or higher-level fusion of multiple sentences is required | Q:Ce ajuta sa faci selfie-uri pe timp de noapte? Sentence: ... Nokia 230 are o camera frontala de 2 MP cu blitz cu LED si o carcasa din aluminiu subtire si eleganta.... Blitz-ul cu LED de pe partea frontala a telefonului te ajuta sa faci selfie-uri pe timp de noapte | 25% |
| Ambiguous | we do not agree with the crowd-worker’s answer, or the question does not have a unique answer | Q:De ce V4 are ergonomie marita? Sentence: ... datorita formei speciale si a finetii date de colturile rotunjite, V4 Viper iti ofera o experienta placuta la atingere ... | 9% |

To test RoITD, fifty random questions were chosen from RoITD and categorized [149]. The details of these categories are provided in Table 3.1. The categorization process provides a deeper understanding of the required reasoning for question answering, similar to SQuAD [167]. These categorization results show that there are always lexical or syntactic differences between the questions and corresponding answers [149].

A summary of the RoITD size is provided in Table 3.2 with additional size and length statistics for question length, answer length, and passage length being shown in Tables 3.3, 3.4, and 3.5, respectively. These summaries shows that majority of questions contain 6-10 words. The majority of the answers contain either 1-5 or 6-10 words. The majority of passages contained 51-80 words. The questions were then divided in eight question type categories—(1) who (i.e., care), (2) what (i.e., ce), (3) when (i.e., cand), (4) where (i.e., unde), (5) why (i.e., de), (6) how (i.e., cum), (7) how many (i.e., cati), and (8) other (i.e.,

Table 3.2: Types of unanswerable RoITD questions [149]

| | Train | Test | All |
|-------------------------|--------------|-------------|------------|
| Number of articles | 4170 | 813 | 5043 |
| Number of questions | 7175 | 2400 | 9575 |
| Average passage length | 52.72 | 61.97 | 55.04 |
| Average question length | 8.08 | 8.12 | 8.11 |
| Average answer length | 13.44 | 7.85 | 9.25 |
| Vocabulary size | 38265 | 18396 | 48821 |

altii). To maintain question diversity, crowd workers were required to manually select the question type. The question distribution is summarized in Figure 3.11.

Table 3.3: RoITD question length statistics. [149]

| Length | Train | Test |
|--------|-------|-------|
| 1-5 | 16.86 | 18.33 |
| 6-10 | 64.04 | 62.54 |
| 11-15 | 17.42 | 16.25 |
| 16-20 | 1.57 | 2.25 |
| > 20 | 0.09 | 0.12 |

Table 3.4: RoITD answer length statistics. [149]

| Length | Train | Test |
|--------|-------|-------|
| 1-5 | 48.62 | 19.45 |
| 6-10 | 23.13 | 23.95 |
| 11-15 | 14.16 | 21.62 |
| 16-20 | 7.67 | 16.29 |
| > 20 | 6.39 | 18.66 |

Table 3.5: RoITD passage length statistics. [149]

| Length | Train | Test |
|--------|-------|-------|
| 1-30 | 10.71 | 3.58 |
| 31-50 | 37.12 | 25.04 |
| 51-80 | 46.23 | 62.33 |
| 81-100 | 5.70 | 6.75 |
| > 100 | 0.22 | 2.29 |

A single NVIDIA Tesla P100 GPU using Google Colaboratory trained the four MRC models using the RoITD [149]. A constant configuration was used for the BERT-based

pretrained MRC models, and the sample initialization parameters were used for the four MRC models where the batch size per gpu is 12, maximum sequence length is 384, document stride is 128, and thread number is 4 [149]. Under these conditions, a learning rate of 3×10^{-5} and training duration of 10 epochs was measured [149]. Standard frameworks including Exact Match (EM) and the F1-score were used to evaluate performance with the results of each BERT-based model being summarized in Table 5.2. As shown in the performance results, RoBERT outperforms both Multilingual BERT and DistilBERT with an EM and F1-score of 35.06% and 53.62%, respectively. This is primarily attributed to the fact that RoBERT is trained exclusively in Romanian using a vocabulary dataset that includes semantics [149]. DistilBERT shows the poorest performance since it is a smaller model than RoBERT and Multilingual BERT. Although Multilingual BERT shows fairly strong performance with an EM and F1-score of 35.48% and 50.94%, respectively, it cannot compete with the advantage that a dedicated Romanian training dataset provides RoBERT. To improve the performance of DL and transformer models, the RoITD quality and size must be improved.

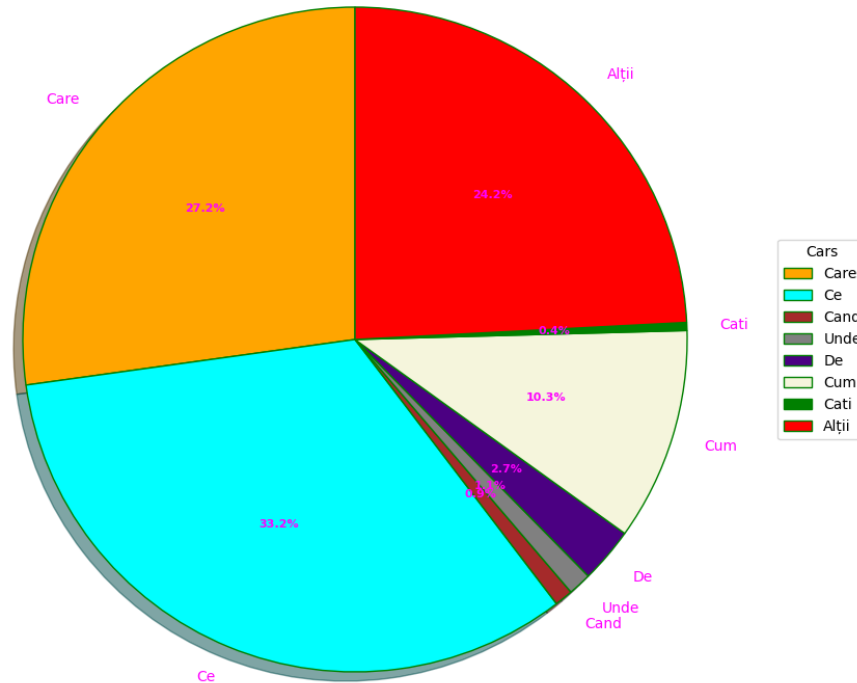


Figure 3.11: Question type distribution.

3.5.2 Romanian Question Classification Task

Even though this dataset is design as machine reading comprehension task, we extend it further to another question classification task. This is inspired by the existing Text REtrieval Conference (TREC) datasets [127]. The TREC Question Classification dataset includes a training set of 5500 labeled questions and a test set of 500. The collection contains 6 level-2 labels and 47 level-1 labels. Each sentence is on average 10 words long, with a vocabulary of 8700 words. Data is gathered from four sources: 4,500 English questions published by USC [104], around 500 hand produced questions for a few unusual classes, 894 TREC 8 and TREC 9 questions, and 500 questions from TREC 10 (which acts as the test set).

Here we extend the RoITD dataset beyond question answering task to formulate a domain classification based on the type of question. Such task is known in question classification. It helps in narrowing down the search area to a particular type of question which helps to extract much better response in QA task. However, this dataset is not generated from human annotators rather automatically using the type of question based on the word. RoITD was developed using human annotators to answer the question and using human annotator again is very costly and time consuming. Hence, we use separating the question based on the words that question contains. In English dataset, the type of question usually depends on the words such as “What”, “Which”, “How”, “Why”, “Who”, “Where”, “When”, and “Whose”. Similarly, we will extract domains of questions in RoITD based on the Romanian version of these words such as “Ce”, “Cum”, “De ce”, “Care”, “Unde”, “Când”, and “Caror”.

Here, we detail the statistics for the question classification. Here, table 3.6 shows the number of questions that covers a particular question type. We can see that among selected type of questions, the dataset does not have “De ce” and only 6 of the questions that contains “Caror”. Hence we neglect these two questions type for their insignificant number. As mentioned in the original datasets, there are altogether 9436 number of questions altogether, only 8704 questions contains these popular question types. Hence our base size of dataset is considered to be 8704. The final selected detail of the question classification dataset is shown in Fig. 3.17. As we see that, the most popular question “Ce” (“What” in English) has the highest samples as expected with 4704 questions. “Care” has the second highest samples with 2724 questions followed by “Cum” with 986 questions. At last. “Unde” and “Când” are the bottom two question types with 128 and 162 respectively.

Table 3.6: Statistics of dataset for different types of questions.

| Question Type | Number of questions |
|---------------|---------------------|
| Ce | 4704 |
| Cum | 986 |
| Care | 2724 |
| De ce | 0 |
| Unde | 128 |
| Când | 162 |
| Caror | 6 |

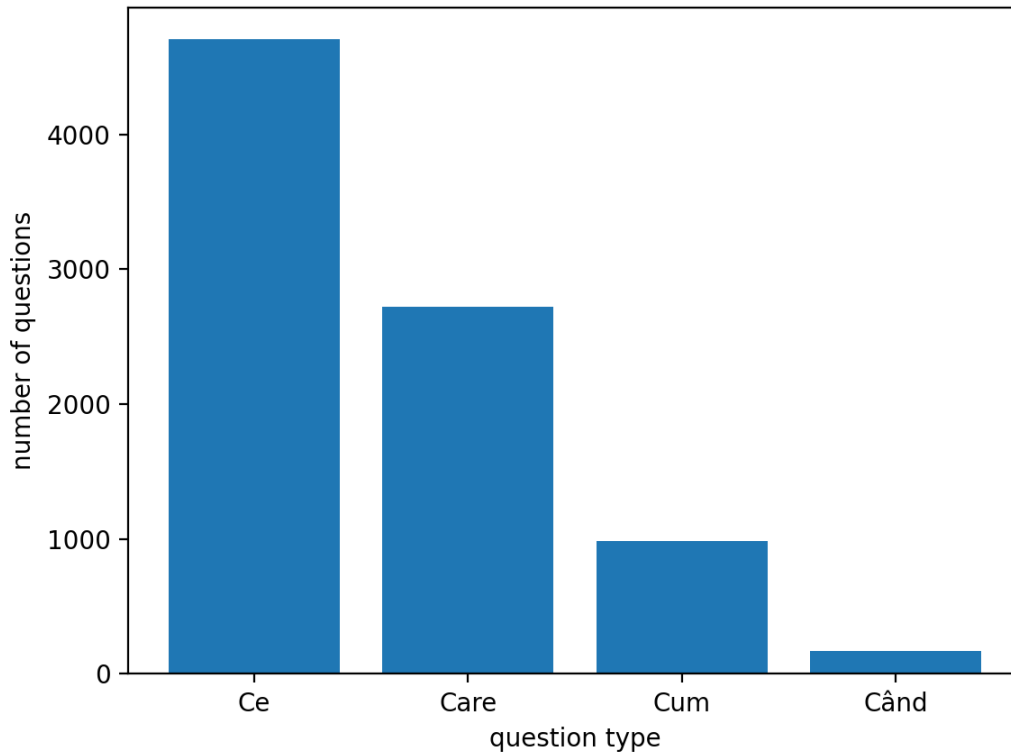


Figure 3.12: Statistics of Question Classification Dataset based on type of questions.

Now as the questions types have been separated, we split the dataset into training and testing various models. We do not specifically separate the training and testing sample instead just use 80:20 random split and run the model using 10-fold cross validation. To establish a simple baseline, we use several models to validate the performance of question classification task. Since, this is an unbalanced dataset, we focus more on macro-F1 score along with the accuracy of the model.

3.5.3 Visual Analysis of the Dataset

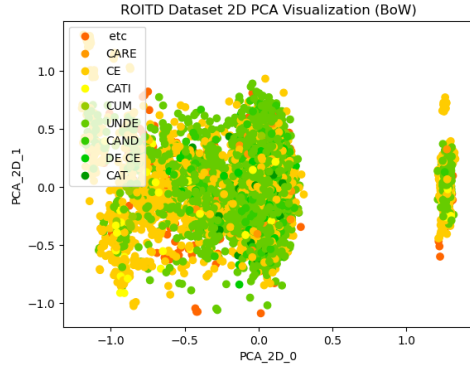
Public and business databases now include gigabytes or terabytes of data on a regular basis due to the exponential growth in data volume. It has long been established that the visual representation of such data is highly valuable for analyzing it, finding correlations, as well as trends and outliers. To do this, it is necessary to somehow transform the high-dimensional data into low-dimensional geometry for visualization. In this section, we demonstrate some visual analysis of the RoITD dataset based on PCA and TSNE. We evaluate both 2D and 3D for bag-of-words (BOW) and tf-idf. One method for collecting high-dimensional data and exploiting the dependencies between the variables to portray it in a more manageable, lower-dimensional form without sacrificing too much information is principal components analysis (PCA). It is one of the simplest and most reliable techniques to accomplish this dimensionality reduction. It is also one of the oldest and has been repeatedly rediscovered in a variety of sectors, earning it the name Karhunen-Loève transformation [15]. Finding the primary causes of data variations is the goal of the PCA transformation approach. A more condensed explanation of the correlations in the data and a deeper comprehension of the underlying features result from the identification of these components. Consequently, it is a potent strategy to draw out broad trends from a data set. Additionally, PCA delivers principal components ranked by their relevance, which makes it a great starting point for data dimension reduction when dealing with multidimensional data. This can be accomplished by ignoring the data set's less important trends in favor of focusing on its key principal components. In many application domains, combining PCA and visual approaches is a typical strategy for high-dimensional dataset analysis [218].

Another data visualization technique is t-Distributed Stochastic Neighbor Embedding (TSNE). It is a method for exploring and visualizing high-dimensional data. To put it another way, TSNE helps you get a sense of how the data is organized in a high-dimensional space. Through a nonlinear dimension reduction technique, it generates a single map that displays the inherent structures in a high-dimensional dataset, including trends, patterns, and outliers. In order to maintain the orientation of the data, PCA obtains the covariance matrix and calculates the eigenvalues and eigenvectors. TSNE, on the other hand, operates in a different manner. Consider a cloud of points in high dimension that we wish to shift to a lower dimension that is also simpler while attempting to preserve the structure of the points, particularly the relationship between the neighbors. It will center a t-Student distribution to each value of the input and use that density to calculate the probability of each point. As a

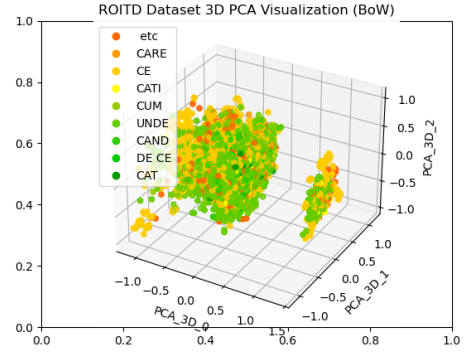
result, conditional probabilities will be created using the Euclidean distance between data points. To maintain the data's structure in the low dimensional map, TSNE will employ a gradient descent method to minimize a cost function. It will produce a decent display of the data by adjusting a few tweakable parameters and improving the objective function. This exhibits a significant reduction in dimensions without sacrificing information.

Figure 3.13 shows the PCA visualization of RoITD dataset based on the type of questions in both 2D and 3D space. Here the input representation of the dataset is in the form of BOW which maps the dataset into various types of questions. Even though we have classified the type of questions based on the type of WH questions, we have not used these WH tags as the features. There is one peculiar behavior in the visualization that is certain samples are at specific gap in the graph. This might not be because of the features type as there are almost all types of WH question present in each cluster. However, the behavior is due to the context these WH questions are used. The contextual difference leads to the shift in distribution of representation. Thus we can observe that the question type are equally distributed across the high dimensional plane making the models to be very precise to differentiate among the classes. This behaviour is also seen in the 3D plot of PCA. Here we can see similar behavior in 3D space as well where the cluster does not seem very distinctive. In addition to this, we also visualize the dataset on PCA using tf-idf as shown in Figs 3.14. PCA using tf-idf separates data into two heavy clusters. In this case we can see that one of the question type that is “Ce” is very distinctive than the rest of the question. The tf-idf demonstrate two heavy cluster one with the question type “Ce”, “Care” and other with “De ce”, “CAND”. One main aspect of such behaviour is tf-idf reattains more information than BOW. It does not only stores the information of frequency but also preserves the frequency of features per document which gives the semantic difference in the higher dimension vector space. This behavior is also seen in the PCA 3D visualization where two types of questions are highly distinctive.

Similarly, fig 3.15 shows the 2D and 3D tsne projection of RoITD dataset based on question type using BOW. We can see that tsne projection shows that all the question type are distributed throughout the vector space without any clustered distinction. This scenario is similarly observed in case of using tf-idf in both 2D and 3D tsne representation as shown in fig 3.16. The feature selection with both techniques such as BOW and tf-idf are unable to distinguish the cluster of question types. Both BOW and tf-idf does not have significant difference based on the visualization of the features. Unlike PCA, TSNE maintains the

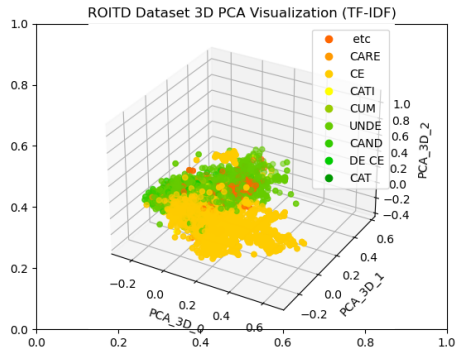


(a)

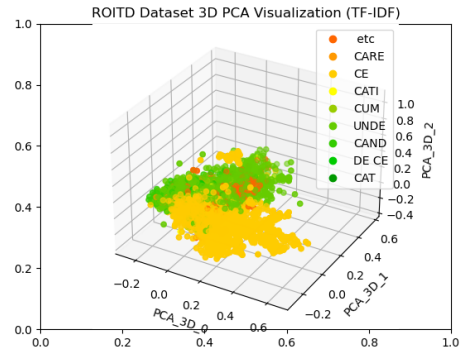


(b)

Figure 3.13: (a) PCA 2D Visualization of BoW. (b) PCA 3D Visualization of BoW.

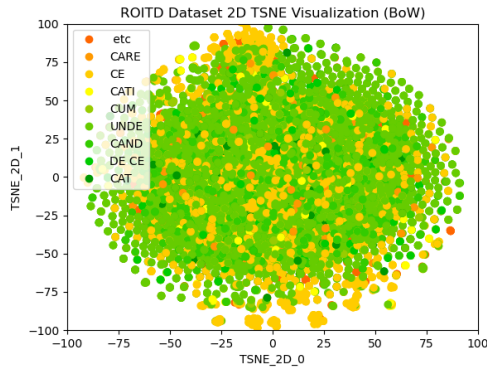


(a)

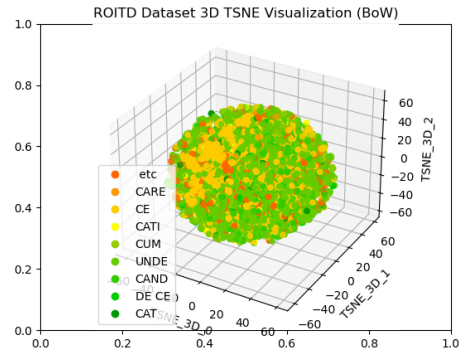


(b)

Figure 3.14: (a) PCA 2D Visualization of TF-IDF. (b) PCA 3D Visualization of TF-IDF.



(a)



(b)

Figure 3.15: (a) TSNE 2D Visualization of BoW. (b) TSNE 3D Visualization of BoW.

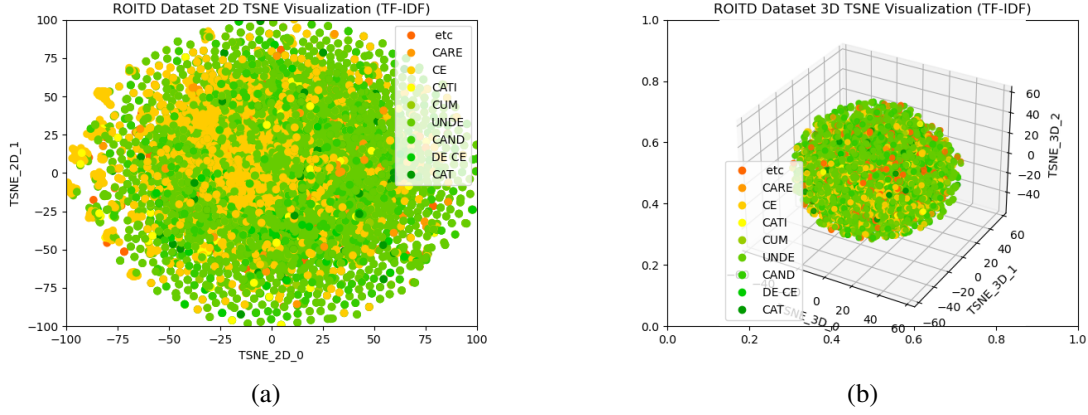


Figure 3.16: (a) TSNE 2D Visualization of TF-IDF. (b) TSNE 3D Visualization of TF-IDF.

local structure of the data by minimizing the Kullback–Leibler divergence (KL divergence) between the two distributions. Also due to the non-linear nature of TSNE, it fails to capture the linear semantic relationship between question type of the context. Similar techniques were used successfully by the authors [49]

3.6 Evaluation

We evaluate the question classification dataset with various models to demonstrate its applicability. We have detailed the process of labeling the dataset in the main file. We used a simple lexicon search rule to assign label to the context. We extracted that there are several types of question based the WH question in English. We understood that more than the type of context/entity, their question types heavily depends on what type of WH aspect is there in the context. Hence, we assigned label based on WH questions and let the model decide the entity that are responsible to make the prediction. We used WH words as the feature to decide the labels. The model itself will determine the important features that makes a particular prediction. This is much simpler way and does not require any kind of additional dictionary. So, the question classification task is not to classify as the entity but as the type of question. This approach basically tells models to find the relevant context by themselves without any additional information. This dataset can be used in two ways: first with the WH question in the context and second by removing the WH questions from the context while preprocessing the context. Our experiment and results demonstrate that most of the models are able to understand the context and perform better except Naïve bayes.

Our intention was to create a self-contained QC task without using any library and only relying on intuition.

3.6.1 Support Vector Machine

For two-group classification issues, a support vector machine (SVM) is a supervised machine learning model that uses classification techniques. SVM models can categorize new text after being given sets of labeled training data for each category. They have two key advantages over newer algorithms like neural networks: greater speed and better performance with a limited number of samples (in the thousands). This makes the approach particularly well suited to text classification issues, where it's common to only have access to a few thousand tagged samples.

A simple example is the easiest way to understand the fundamentals of Support Vector Machines and how they work. Let's say our data contains two features: x and y . The hyperplane (which in two dimensions is essentially a line) that optimally separates the tags is produced by a support vector machine using these data points. This line serves as a decision border, and anything that falls on one side of it will be divided into two classes.

3.6.2 Naive Bayes

The Naive Bayes algorithm is a probabilistic machine learning technique that may be applied to a wide range of classification applications. Filtering spam, categorizing documents, and predicting sentiment are examples of common applications. The term naïve refers to the assumption that the features that make up the model are unrelated to one another. That is, changing the value of one feature has no direct impact on the value of the other features employed in the algorithm. Because it is a probabilistic model, the program may be quickly coded and predictions made. Quick in real time. It is scalable as a result, and it is traditionally the algorithm of choice for real-world applications (apps) that must reply to user requests instantly.

3.6.3 Decision Tree

Decision Tree is a versatile tool that can be used in a variety of situations. Both classification and regression problems can be solved with decision trees. The name implies that it use a tree-like flowchart to display the predictions that result from a sequence of feature-based splits. It begins with a root node and finishes with a leaf decision.

- **Root Nodes:** It is the first node in a decision tree, and it is from this node that the population begins to divide according to numerous characteristics.
- **Decision Nodes:** Decision Nodes are the nodes that result from separating the root nodes.
- **Leaf Nodes:** Leaf nodes or terminal nodes are nodes where further splitting is not possible.
- **Sub-tree:** A sub-section of this decision tree is referred to as a sub-tree, just as a small portion of a graph is referred to as a sub-graph.
- **Pruning:** is cutting down some nodes to stop overfitting.

3.6.4 Tsetlin Machine

Tsetlin Machine is a new categorization method based on a team of Tsetlin Automata that manipulates expressions in propositional logic (TA). TA is a deterministic automaton with a fixed structure that learns the best action from a set of suggestions from the environment. Each input bit in Tsetlin Machine is represented by two TAs, TA and TA'. The input sample's original bit is controlled by TA, while its negation is controlled by TA'. Each TA represents a single literal. A literal denotes an input bit or its inverse. Any TA used by a Tsetlin Machine has two actions, each of which has $2N$ states. total. Action "exclude" is chosen for states 1 to N , while action "include" is chosen for states $N + 1$ to $2N$. A TA conducts "include" or "exclude" based on the current state for each iteration. As a result, a reward or punishment is triggered. If the TA receives a reward, it moves to the deeper side of the action, whereas if it receives a penalty, it moves to the center and eventually jumps to the other side.

Tsetlin Machine controls a decentralized team of TAs using a novel game theoretic strategy. By incorporating or eliminating specific literals, this technique aids the TAs in learning an arbitrarily complex propositional formula. The included literals, in particular, form clauses through the operation of conjunction. After training, each sentence is anticipated to catch a sub-pattern. Let us consider the input feature as a vector with a vocabulary size of n words, which is represented in BOW as $X_s = [x_1, x_2, x_3, \dots, x_n, \dots, x_{2n}, x_{2n+1}, x_{2n+2}, \dots, x_{2n+12}]$ with $x_k \in \{0,1\}$ and $k \in \{1, \dots, 2n+12\}$. Here, (x_{2n+1}) to (x_{2n+6}) represent LOC_{vec}^1 and LOC_{vec}^2 . Similarly, (x_{2n+7}) to (x_{2n+12}) represent SC_{vec}^1 and SC_{vec}^2 . Let q be the number of classes ($q = 3$ in ABSA task: positive, neutral and negative). If a pattern has m sub-patterns, the pattern can be captured using $q \times m$ conjunctive clauses C_i^j , $1 \leq j \leq q$, $1 \leq i \leq m$:

$$C_i^j = \left(\bigwedge_{k \in I_i^j} x_k \right) \wedge \left(\bigwedge_{k \in \bar{I}_i^j} \neg x_k \right), \quad (3.1)$$

where I_i^j and \bar{I}_i^j are non-overlapping subsets of the input variable indices, $I_j^i, \bar{I}_j^i \subseteq \{1, \dots, 2n+12\}$, $I_j^i \cap \bar{I}_j^i = \emptyset$. The subsets decide which of the input variables take part in the clause, and whether they are negated or not. The indices of input variables in I_j^i represent the literals that are included as is, while the indices of input variables in \bar{I}_j^i correspond to the negated ones. Among m clauses in each class, clauses with odd indexes are assigned positive polarity (+) whereas those with even indexes are assigned with negative polarity (-). The clauses with positive polarity vote for the target class and those with the negative vote against it. A summation operator aggregates them by subtracting the total number of negative votes from positive votes, as shown in Eq. (3.2).

$$f^j(X_s) = \sum_{i=1,3,\dots}^{m-1} C_i^j(X_s) - \sum_{i=2,4,\dots}^m C_i^j(X_s). \quad (3.2)$$

For q number of classes, the final output y is given by the argmax operator to classify the input based on the highest sum of votes, as shown in Eq. (3.3).

$$y = \operatorname{argmax}_j (f^j(X_s)). \quad (3.3)$$

3.6.5 Results

The performance of these above-mentioned models are explained in this section. Table 3.7 shows the macro-F1 and micro-F1 for all selected baselines. As we can see that SVM performs really good with micro-F1 reaching 95.23% and macro-F1 reaching 85.06%. This means that even the dataset is unbalanced, the model performs good on the task. Decision tree also performs quite similar to the SVM in terms of macro-F1 and micro-F1. Additionally, on of the most powerful linear model i.e., logistic regression outperforms other models by achieving Macro-F1 of 87.45% and 95.78%. However, Tsetlin Machine surpasses all of them by reaching macro-F1 to 88.61% and micro-F1 to 95.98%. On the other hand, Naive Bayes performs poorly on this particular task with lowest macro-F1 and micro-F1. This baselines shows that even though the dataset is unbalanced, it is well learnt my majority of the models.

Table 3.7: Performance of various models on question classification task.

| Models | Macro-F1 | Micro-F1 |
|---------------------|----------|----------|
| SVM | 85.06 | 95.23 |
| Naive Bayes | 68.55 | 87.01 |
| Decision Tree | 85.45 | 95.34 |
| Logistic Regression | 87.45 | 95.78 |
| Tsetlin Machine | 88.61 | 95.98 |

3.6.6 Discussion

Since, the comparison here are done only with relatable models and the base of comparison is interpretability. Interpretability is an important aspect of modern NLP and in the application of chatbot. Many scholars began to investigate ways to explain the intelligent system, with the primary goal of creating more understandable representations to lessen the complexity of difficult rules. AI systems today, on the other hand, are not the same as rule-based systems in the past. Because of the growing number of factors and procedures, the model gets more complicated, making it more difficult to explain the model's choice. So yet, there hasn't been a consensus on a definition. Doshi-Velez and Kim [68] define interpretability (or explainability) as the ability to explain or communicate something to a person in a way that they can comprehend.

Simple models, such as linear classifiers, are often easier to grasp, but a sophisticated

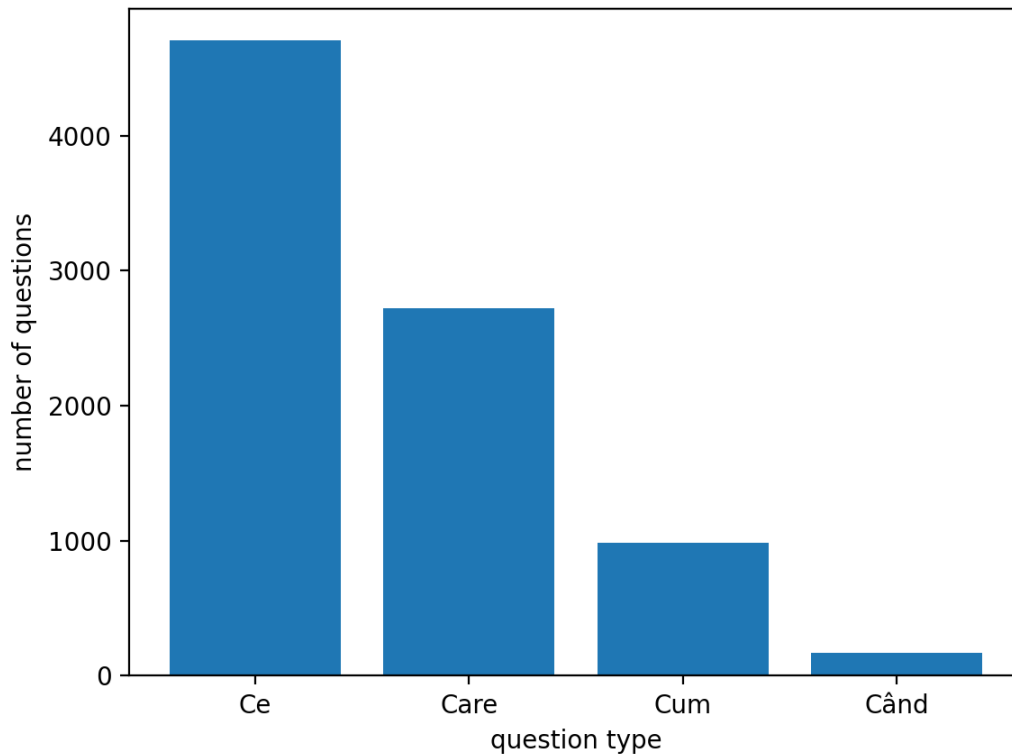


Figure 3.17: Statistics of Question Classification Dataset based on type of questions.

model, such as a deep neural network, is challenging to comprehend due to its layer-wise structure and nonlinear computing. In this section, we define the explainability of an AI system as the ability to explain the reason why it makes the decision in a human-understandable way, alternative approaches of it. There's a natural question that arises. Why do we need to be able to explain things? The causes may be divided into four categories: user trust, system change, system learning, and moral and legal concerns.

- **Trust from users:** A black box is the sophisticated machine learning model. Ordinary users have no idea how it makes judgments or how to ensure that they are right. The users may doubt the model's reliability if the model cannot explain itself. However, once the model can explain itself, users are better able to understand why the model appears to make the right or wrong decision, so that users can have more confidence in the system.
- **Modification of the system:** The problem with complex models such as deep neural networks is that they have too many operations and parameters. If developers do not

know why an AI system performs poorly, they cannot debug it. It should therefore be easier to improve an AI system if the developer is able to easily modify it.

- **Learn from the system:** The AI system can sometimes teach us new insights. For example, the system has outperformed human Go players and performed strategies that we had never imagined. With explainable AI models, we can obtain some new insights or knowledge.
- **Moral and legal issues:** Since artificial intelligence has gradually entered our everyday life, legal issues related to AI's wrong decisions have received relatively little attention until recently. Hence, our reliance on black-box models makes it difficult to find a perfect solution to these legal issues.

Humans consider interpretable models easier to understand without explicit explanations. For instance, rules-based methods such as decision trees have nodes and branches to clarify their reasoning when making decisions. For humans, tracing from the leaf to the root is a more intuitive and straightforward way of understanding a decision. In order to overcome the difficulty of constructing high-accuracy interpretable trees, Bayesian Rule List introduced a Bayesian framework to rule-based methods. As the number of nodes and rules grows rapidly, it becomes harder to understand the classifier as a whole. There are two modes of explainability Local and Global:

Local Explainable Methods

If the explainable method can generate explanation for the model around a given input point, then we call this a local explanation. The behaviour of a model is usually explained by using some inputs and their accompanying predictions as examples. Based on whether the explanation is derived directly from parameters or structures in the model, the local explanations are divided into two types, model-unaware explanations and model-aware explanations.

- **Model-unaware explanations:** Generally, model-unaware explanations are derived from sensitivity. A squared partial derivative is used in image classifiers to calculate the saliency score based on a given input image, highlighting the most sensitive part that gives the classification the strongest spatial support. Similarly, Li et al. [126]

compute the first-order derivative as a salience score for a unit from different RNN classifiers for sentiment analysis, and generate heatmap matrices as an explanation of which dimension in the embedding vector or which word is crucial for predicting the outcome. Despite the effectiveness and intuitiveness of generating explanations based on sensitivity, complex models have a high degree of non-linearity, resulting in noisy explanations. In spite of their advantages, such as the ability to compare different deep models easily, this method has difficulty understanding the global situation and having inconsistent explanations for different samples.

- **Model-aware explanations:** The parameters of the model are often used to derive model-aware explanations. Convolutional neural networks visualize important pixels in input images that correspond to classification using gradients from the final convolutional layer. In addition, Layer-wise Relevance Propagation (LRP) is used to transfer the relevance score from the output layer to the input layer of CNNs using linear sum-pooling and convolution or simple multiple perceptron. This approach might be used to calculate the contribution of each input variable. Certain efforts try to show how the deep model learns, to visualize the RNN’s hidden units, and to discover some learning processes for how an LSTM language model learns.

Global Explainable Methods

We consider an explanation to be global if it is generated for the whole input space or as an overview of how the model acts. We also address global explainable methods from model-unaware and model-aware approaches, which are similar to the category of local explanations.

- **Model-unaware explanations:** There hasn’t been much research on how to produce model-unaware explanations worldwide in a model-agnostic manner. Ribeiro, Singh, and Guestrin [50] [172] construct global explanations by presenting users with a collection of sample local explanations one at a time. This strategy is prone to failure when there is an abundance of training data and users are unable to recall a large number of representative local explanations in order to develop a global view.
- **Model-aware explanations:** One approach is to consider the activity of each neuron in response to an input as a semantic property of the concepts represented by this

input text. They connect neurons with human-understandable ideas throughout this dissecting process to determine how effectively a notion is represented by a single unit.

Above-mentioned points clearly indicate the importance and attempt of explainability in the field of NLP. Since, our task deals with a feasible QA model for chatbot application, building a prerequisite question classification models is as important as it gets. Since, the user asks for a definite explanation of the prediction and there is a massive bottleneck to explain the big language models, we attempt to provide a simple logical explanation of question classification. This basically reduces the gap between explainability and accuracy to some extent. Hence, we used all the simple explainable models for this task. Among the selected baselines, logistic regression is arguably simple and interpretable model along with decision tree.

Decision tree being one of the great model for explaining the model's prediction. It uses root and nodes to explain its decision with a very good accuracy as shown in Table 3.7. However the limitation of this model comes into the picture when the size of tree grows due to big datasets and features. This creates bottleneck for scalability. Another approach for easier explainability is Logistic Regression which performs great compared to other simpler machine learning models. The explainability of the model is visualized by using the weights of the features impacting the prediction. However, this mathematical weight coefficient only express the impact of each features which can not be comprehend to humans to make a logical interpretation. Human tends to understand logically rather than evaluating mathematical weights. Hence, in order to fill this gap, we used Tsetlin Machine that not only performs better than all selected models but also offers human level interpretation. TM uses clauses to learn pattern from the combination of features known as propositional logic. Such propositional logic are easier to understand for human. This is because propositional logic are combination of features and their negation in conjunctive form which is much easier to comprehend in compared to mathematical weights and trees structure.

3.7 Outlier Detection

Machine learning (ML) models offer immense flexibility through their ability to learn from many types of data including both structured (e.g., client information and tabular data) and semi-structured (e.g., text data and images) [107]. However, this learning flexibility requires that the model be able to deal with outliers, which in this case are data points that are substantially different from the training dataset [97]. Since these types of outliers are typically encountered after model deployment, the model must be able to detect them while running to prevent performance reduction [21]. Unfortunately, outliers are often present within inlier data clusters 3.18, making them difficult to identify despite the fact that they follow inconsistent patterns compared with the inliers [107].

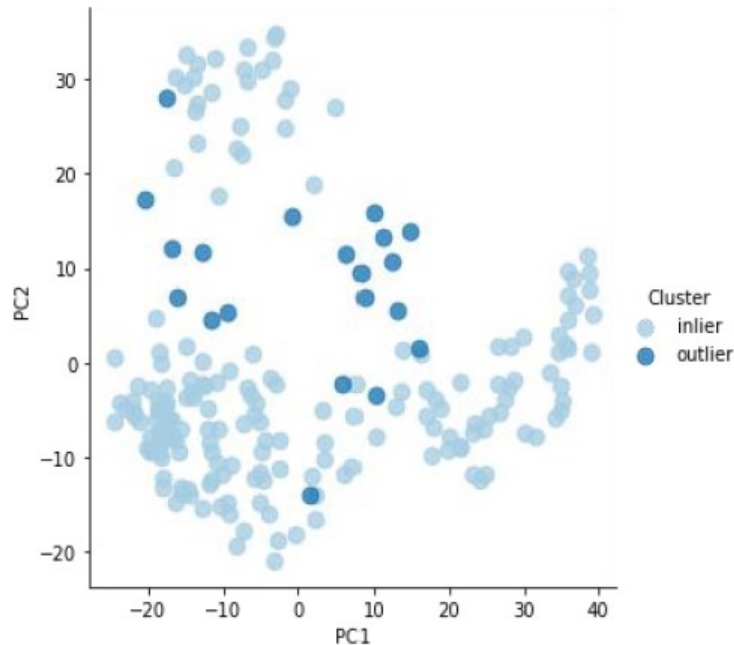


Figure 3.18: Outlier and inlier distribution of the MNIST dataset. [107]

Generally, outlier detection methods generally fall into one of three categories—parametric, non-parametric, and semi-parametric [33,36]. Parametric models use pre-selected distribution models to fit data, making them the least flexible with limited applications [204]. Non-parametric models are more flexible with greater autonomy similar to neural networks. Semi-parametric models combine the speed and complexity of parametric methods with the flexibility of non-parametric models, making them a popular option for a variety of applications such as Gaussian mixture models [75,77,110,147].

Of the available outlier detection methods, K-means, Density-Based Spatial Clustering of Applications with Noise (BDSCAN), Gaussian Mixture Models (GMM), and One Class-Support Vector Machine (OC-SVM) are among the most commonly used. K-means uses partition-based clustering to group data according to similarity where similarity is defined by distance [223]. Clusters are centered with a radius defined by the farthest point in the cluster, making this method ideal for circular data groupings [223]. K-means relies on hard classification where data points are defined by which cluster they belong to, and the probability of a given data point belonging to any given cluster is not calculated [223]. DBSCAN is an unsupervised algorithm that uses density-based clustering to identify high-density regions, providing identification capability for arbitrary-shaped clusters and clusters containing noise caused by outliers. GMMs use a similar clustering procedure as k-means, but includes modifications such as variance inclusion that allows for calculation of the distribution shape [137]. As such, GMMs are able to handle a wide variety of cluster shapes, including very oblong ones [137]. GMMs also include soft calculations, where the probability of a data point belonging to any of the possible clusters is defined [137]. SVMs rely on hyperplanes within multi-dimensional space to separate observations by class [23]. OC-SVM is used for datasets where all the data belongs to only one class, allowing the algorithm to learn about the structure of inliers to improve outlier identification [23].

Recent literature has established that deep learning methods outperform machine learning methods for outlier detection applications. A new Generative Adversarial Networks (GAN) with Variational AutoEncoders (VAE)-based generative deep learning model that separates inliers and outliers using uniform distributions achieved by variational autoencoders has been introduced [107]. This method targets zero-shot outlier detection where the applied method cannot access outlier data points from the training dataset. This method learns the inlier underlying structure to improve outlier identification. A OC-SVM trained on the training data loss values is used to form the final decision [46]. Three primary losses are obtained using this model (i.e., reconstruction loss, KL-divergence, and discriminative loss) using the generative and adversarial portions of the model. Predictions are then made by wrapping these losses using a one-class SVM. An evaluation of this model performance along with other state-of-the-art methods for outlier detection, including K-means, DBSCAN, GMM, and OC-SVM, confirms that this new VAE-GAN-based model outperforms the other models for all datasets.

For the task, we have asked our crowd-source workers to label each question-answer

pair with 1 if contain at least one grammar mistake, according to their understanding. This generated a total of 709 positive question-answer pairs, from a total of 9157. The label data set is publicly available [148].

3.7.1 VAE-GAN Based Zero-shot Outlier Detection

The Encoder (E), Decoder/Generator (D/G), and Discriminator (D) form the primary components of the VAE-GAN-based model. The dataset being used is described by $X = (x_i, y_i)_{i=1}^N$ where N represents the number of samples, x_i is the data sample, and y_i is the optional class of the i^{th} sample for which $y_i \in A$, $A = \{0, 1\}$ is for the purpose of evaluation [107]. In this case 0 represents an inlier while 1 represents outliers. The Convolutional Neural Network/Multi Layer Perceptron (CNN/MLP) Encoder with W^E weights provides latent representation of the inliers and outliers. These representations can then be applied as random seeds to generate data, defined as [107]:

$$\mu, \sigma = E(x); z \sim N(\mu, \sigma) \quad (3.4)$$

where $z \in Z$ are the extracted features that represent the data state. For an input sample of x^{in} inliers, the Encoder output is $z^{out} = E(x^{out})$. The Encoder output is then fed to the Decoder/Generator. This is the intersection between the GAN and VAE portions of the model. As z is sampled from the normal distribution provided by the Encoder, z is fed through the Decoder to reconstruct the original input samples as follows [107]:

$$\hat{x} = D(z); z \sim N(\mu, \sigma) \quad (3.5)$$

where \hat{x} is the reconstructed input sample contained in the latent representation. This is additionally referenced by $x^{gen} = G(z)$. The Discriminator forms the final portion of the model. The Discriminator is a binary Feed Forward Neural Network binary classifier that determines the data point type (i.e., real or generated) as shown by the following equation [107]:

$$a = D(w); w \in (X^{gen} \cup X^{real}); a \in A; A = \{0, 1\} \quad (3.6)$$

where X^{gen} is the collection of the generated samples produced by the Generator while X^{real} is the actual dataset. The standard Evidence Lower Bound (ELBO) is used to train the neural networks, along with adversarial generative loss for the GAN component. Three loss functions are used for the VAE-GAN-based model. The first is the distance function between the input and reconstructed samples, known as the Reconstruction Loss, L_R , and described by the following equation [107]:

$$L_R = d(x, \hat{x}); x \in (X^{gen} \cup X^{real}) \quad (3.7)$$

where $d(x, \hat{x})$ is the Mean Squared Error in the case of images and the cosine distance in the case of vectors. The KL-divergence, L_{KL} [118] is the next loss function where the variational approximation $q(z|x(i)) = N(\mu, \sigma)$ is represented by the Encoder output for each sample, and the true distribution $p(z) = N(0, 1)$ of the latent variable, z , serves as the normal distribution of mean = 0 with standard deviation = 1. This loss function is described by [107]:

$$L_{KL} = KL(q(z|x(i)), p(z)) \quad (3.8)$$

The discrimination probability, L_D is the final loss function and used for anomaly detection. This loss function is provided by the Discriminator directly using the original sample as opposed to the reconstructed or generated samples and is described by [107]:

$$L_D = D(x); x \in X^{Rel} \quad (3.9)$$

These three losses are combined in a 3-feature dataset that is fed into a One-Class SVM to classify samples as inliers or outliers.

The methods were evaluated using three datasets—MNIST [123], CreditCardFraud [185], and KDDCUP99 [26]. The CreditCardFraud dataset serves as an example of datasets containing a small number of outliers and a large number of inliers. In this case, the loss functions of the VAE-GAN-based algorithm are able to identify the outliers and inliers without the need for the OC-SVM. Table 3.19 compares the outlier detection performance of the tested models using F1-scores [107]. In this case, DBSCAN shows the worst per-

formance (F1-score = 0.5542) while GMM shows the strongest (F1-score = 0.694), followed by OC-SVM then K-means. Table 3.20 shows the detailed performance results of the VAE-GAN-based method on the CreditCard-Fraud dataset, along with the performance using each of the loss functions individually [107]. The F1-score of this model is substantially higher than the other tested models (F1 = 0.8376), and the Reconstruction loss-only test produced the highest performance compared with the other two loss functions (F1 = 0.8165).

| Method | F1-score |
|---------|-----------------|
| K-means | 0.6513+-0.022 |
| BDSCAN | 0.5542 +-0.063 |
| GMM | 0.694+-0.015 |
| OC-SVM | 0.6712 +- 0.003 |

Figure 3.19: Outlier detection performance of various models on CreditCard-Fraud raw dataset. [107]

| Model Input | F1-score |
|---------------------|-----------------|
| KL-divergence only | 0.7551+-0.04 |
| Reconstruction Only | 0.8165 +-0.064 |
| Discrimination Only | 0.6632+-0.029 |
| All combined | 0.8376 +- 0.072 |

Figure 3.20: Zero-shot outlier detection performance of VAE-GAN-based method on CreditCard-Fraud dataset. [107]

Tables 3.21 and 3.22 compare the VAE-GAN model performance compared with other state-of-the-art models for the MNIST and KDDCUP99 datasets, respectively [107]. MNIST is an image-based dataset while the KDDCUP99 dataset allowed for comparison with other GAN-based models. In the case of the MNIST dataset, the VAE-GAN-based model outperformed the other models with the exception of the lowest outlier percentage (10%), where the VAE-GAN-based model is only outperformed by the $D(R(x))$ model. In the case of the KDDCUP99 dataset, the VAE-GAN-based model produces the second-highest F1-scores for zero-shot outlier detections. Although DAGMM-NVI outperforms the VAE-GAN-based model, the scores are nearly equivalent (0.93 compared with 0.92).

The VAE-GAN-based model was also evaluated for supervised outlier detection using the CreditCard-Fraud dataset [107]. A summary of these results is provided in Table 3.23.

| Model | 10 | 20 | 30 | 40 | 50 |
|--------------|-------------|-------------|-------------|-------------|-------------|
| LOF [27] | 0.91 | 0.83 | 0.72 | 0.68 | 0.56 |
| D(x) [29] | 0.91 | 0.89 | 0.86 | 0.85 | 0.83 |
| DRAE [28] | 0.94 | 0.91 | 0.87 | 0.82 | 0.76 |
| D(R(x)) [29] | 0.96 | 0.92 | 0.92 | 0.91 | 0.89 |
| ours | 0.95 | 0.93 | 0.93 | 0.94 | 0.91 |

Figure 3.21: F1-scores of VAE-GAN based method compared with other zero-shot outlier detection methods for the MNIST dataset. [107]

| Method | F1-score |
|-----------------|-------------|
| OC-SVM | 0.79 |
| DAGMM-NVI [30] | 0.93 |
| AnoGan-FM | 0.88 |
| AnoGan σ | 0.78 |
| Ours | 0.92 |

Figure 3.22: F1-scores of VAE-GAN based method compared with other zero-shot outlier detection methods for the KDDCUP99 dataset. [107]

Once again, the Reconstruction Loss-only produces the highest F1-score compared with the other two loss functions, and the overall model produces a high F1-score of 0.9131.

| Model Input | F1-score |
|---------------------|----------|
| KL-divergence only | 0.7713 |
| Reconstruction Only | 0.8789 |
| Discrimination Only | 0.582 |
| All combined | 0.9131 |

Figure 3.23: VAE-GAN based method supervised outlier detection performance for the CreditCard-Fraud dataset. [107]

Overall, it is clear that the addition of a VAE to traditional GAN models greatly improves algorithm performance for outlier detection applications. This VAE-GAN model feeds the extracted loss functions for each component into an OC-SVM to classify data points as inliers or outliers. This method performs well for a wide variety of datasets, including image and high-dimensional tabular datasets. Additionally, the model maintains

strong performance for variations in the amount of outliers contained within the data.

3.7.2 Results ROITD outlier detection

Natural language processing (NLP) has been developed for many advanced artificial intelligence (AI) applications by enabling natural language comprehension, interpretation, and manipulation. However, in order for these applications to succeed, a high-quality corpus is needed. This is no trivial task since data access itself can pose a challenge, and the corpus must maintain proper grammar and any other required formatting to run the NLP model.

An NLP library corpus is a collection of natural language text or audio that has been organized into a well-formatted dataset. Ideally, this corpus is fairly large. The corpus is then used to train AI and machine learning (ML) NLP models. The corpus also requires a specific formatting, including proper grammar, to provide the model with the ability to process natural language along with contextual data. Once a properly structured corpus is provided, the model can complete advanced applications including question answering (QA) or summarization tasks.

Annotation is used to enhance and modify a corpus with advanced information, such as part-of-speech tagging, word senses, or word meanings. By annotating the corpus, a model can be trained to recognize various speech patterns. Whatever the model learns about natural language from the corpus can then be translated to new, unannotated data to deliver impressive results. As previously mentioned, one important aspect of the corpus is that it be sufficiently large. This provides a statistically robust natural language sample for the model to learn effectively. However, this cannot come at the expense of data quality. The presence of any improperly formatted data within the corpus can dramatically reduce the final model performance. It cannot be emphasized enough that the quality level of the corpus is by far the most important factor.

That being said, a larger high-quality dataset will produce stronger results, although it is possible for a corpus to be too large. A corpus that is too big can slow down the model and produce inaccurate results. For instance, too much data can produce overfitting where a model learns from the corpus details and noise too well. This reduces model performance once new data is introduced. The corpus size controls manageability and practicality of

data collection since a substantial time investment is required to transcribe and annotate the thousands or even millions of words required to produce a properly annotated, clean corpus.

The most important aspects of creating a high quality corpus are accuracy, completeness, and timeliness. The values and metadata included in the corpus must be checked for accuracy. This will provide efficient and effective task performance by the model. This includes cleaning the data by removing any errors including outdated, incorrect, irrelevant, or duplicate data. Data cleaning ensures the data quality. Next, the corpus data must be checked for completeness to be sure that there are no gaps or other missing information that will prevent the model from learning natural language insights. Finally, the corpus needs to be continuously updated to be sure the data maintains relevancy. It is also important to include balance within the corpus. This means that the data collection process should be streamlined and structured to ensure relevancy. The data balancing step is dependent on the final application.

Grammar itself can be used to teach important language tools to models, making this component especially important in a NLP corpus. For instance, scheduling information is a common inquiry input from users, but this information is generally contained within a complex relationship between prepositional objects and parent verbs. If the NLP corpus does not contain accurate grammar, the model can experience several failures including learning incorrect language rules, sourcing incorrect information, and producing outputs that cannot be understood by humans. This situation extends to many other language examples where grammar is used to define context or even meaning within a sentence. Ensuring that the grammar within a training NLP corpus is correct will ensure the most accurate model results by teaching proper sentence structure that will be translated to any output produced by the model. This is also a critical aspect in the production of text than can be understood by humans, which is often the main objective of NLP tasks.

We proceed to compare Alphagan [107] with more established methods such as K-means, LOF, OC-SVM and DBSCAN and the results are displayed in Table 3.8 3.25 3.24.

Regarding the methods, we note that the first most efficient method is Alphagan, with an F1-score of 67.8 % and an accuracy of 96%. Out of 9157 Questions/Answers 8803 were correctly predicted. And out of 706 positive Q/A, 352 were predicted as outliers.

| Method | F1 |
|----------|--------|
| Alphagan | 0.6786 |
| LOF | 0.4865 |
| K-means | 0.4823 |
| BDSCAN | 0.4799 |
| GMM | 0.4151 |
| OC-SVM | 0.4814 |

Table 3.8: Performance of different outlier detection algorithms on RoITD dataset .

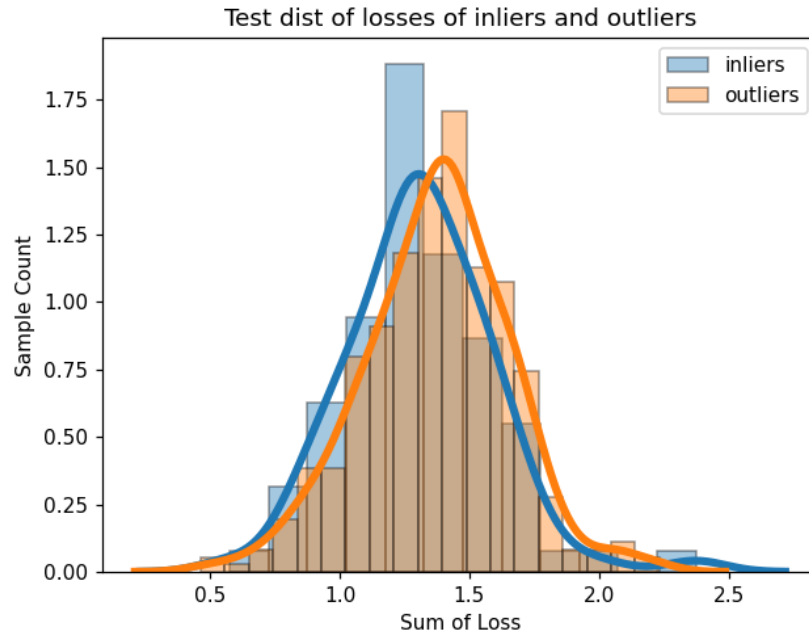


Figure 3.24: ROITD Alphagan Test Loss Distance.

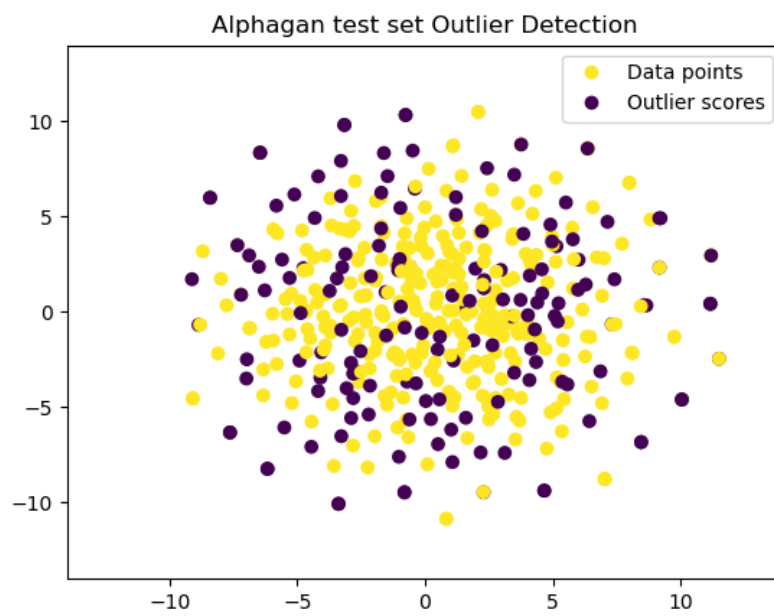


Figure 3.25: Alphagan Vizualization.

Chapter 4: MACHINE LEARNING ADVANCEMENTS

This chapter provides an overall literature review of basic machine learning (ML) concepts. We then move on to a specific area of ML called DL. After these basic concepts have been discussed, we explain the BERT model in detail. The BERT model includes transfer learning like transformers, or self-attention mechanisms, which are elaborated on to understand their role in the proposed project.

4.1 Machine Learning

At first we explain the basic concept of Machine Learning which is the base for recent Natural Language Processing. We start with the concept for machine learning leading to more sophisticated model in the later part.

Arthur Samuel first coined the term "machine learning" around the late 1950's [182]. Samuel is a researcher in Artificial Intelligence who developed statistical modeling techniques with the capability of learning patterns from sample data. When the model is perfectly trained, it is capable of performing classification, prediction, and decision when new data (excluded from the training sample) is passed. Multiple methods can be implemented to categorize the learning process including reinforcement learning, unsupervised learning, and supervised learning [146].

4.2 Deep Learning

Deep Learning is one of the most popular choices for recent advances in NLP tasks, which consist of various important elements. We here explain the basic concept of deep learning and the evolution of their variants. Deep Learning (DL) is a ML technique that uses artificial neural networks to simulate the structure and functions of human brains [63]. DL is also considered to be a structured and hierarchical learning that uses huge number of hidden layers in non-linear processing. DL helps to extract features from the unstructured data and generate various form of representation. DL is also defined as a hierarchical probabilistic model that recognizes pattern with several layers of abstraction. Knowing that there are multiple data samples, these multi-layer neural networks are able to decompose a

problem into various smaller and more manageable abstractions. Compare to other models of ML, DL performs significantly better but also requires intense parameter tuning to achieve high performance. This model has opened a vast area of research to develop newer models based on neural networks or to fine tune parameters for better performance. Some of areas where DL has excelled include NLP, speech recognition, and computer vision.

4.2.1 Neural Networks

The main approach of DL is the Deep Neural Network (DNN), which is very different from shallow neural networks . DNN consists of several hidden layers (usually more than 3 hidden layers) and can be trained by both supervised and unsupervised methods. These models are generally composed of nodes in a particular way that somewhat replicates the interactions between biological neurons. In essence, it acts like a neuron to pass information based on input to create the simplified mathematical model, usually referred to as the perceptron [175]. Each neuron in the network receives multiple numeric values as input. These inputs are assigned with weights w that gives importance from the input to the computed output at the node.

4.2.2 Recurrent Neural Networks

Unlike traditional feed-forward neural networks (FNN), Recurrent Neural Network (RNN) [131] has the function of remembering the previous information and applying it to the current input. Although CNN can effectively for spatial data, it is not an ideal method for processing sequential or temporal data. The RNN architecture is especially important for NLP, because the entire text consists of a sequence of words, and, therefore, a sentence. Repetitive RNN connections can be visualized as unfolded as in Fig 4.1. Here, the original layer will be copied as many times as necessary to cover all the timestep of the entire sequence.

Sequential data is usually split by time, and RNN accepts input data mapped to data at a given time step. Each time the output of a step is forwarded back to the network, it will ensure the previous state is recorded, which will affect the subsequent step results, so it is called “recurrent”. Persistence information allows the network to process the next inputs,

including previous inputs. Basic recurrence can be expressed as

$$h_t = f_h(h_{t-1}, x_t) \quad (4.1)$$

where h_t and h_{t-1} are the new hidden state and previous hidden state, respectively, f_h is the activation function, and x_t is the input at time step t .

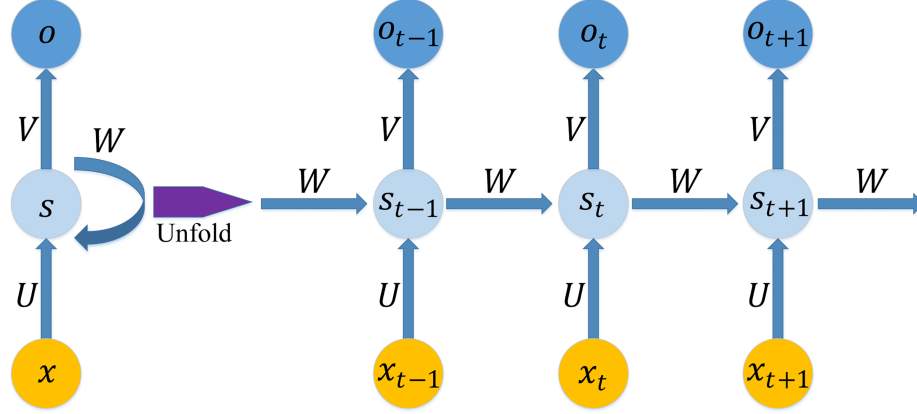


Figure 4.1: Unfolded recurrent Neural Network [124].

4.2.3 Long-Short Term Memory

The traditional RNN does not support long sequences of data. In addition, because the gradient will disappear or explode, RNN training may be difficult, causing problems when propagating back at large temporal intervals [37]. To manage the long-term dependence on training data, a new network architecture with learning gates is used, which is known as LSTM. LSTM contains memory blocks with memory cells called gates in the repeated hidden layer, as shown in Fig 4.2.

These gates help regulate information flow inside the LSTM unit. The usual gates included are an input gate i_t , an output gate o_t and a forget gate f_t . In addition, the LSTM maintains two hidden states during each time step. First, the hidden state h_t already exists in the traditional RNN. Second, the state of the cell state c_t acts as a memory that interacts with the gates. The LSTM training process mainly focuses on learning when to initiate an activation in internal state of its cell and when to activate the output. The main idea of LSTM is not only to evaluate the influence of each word in the sequence on the hidden state, but also to include unimportant words that are safe to "forget". In addition to these mechanisms, the cell state transfers the gradient in both directions in a cleaner flow, thereby

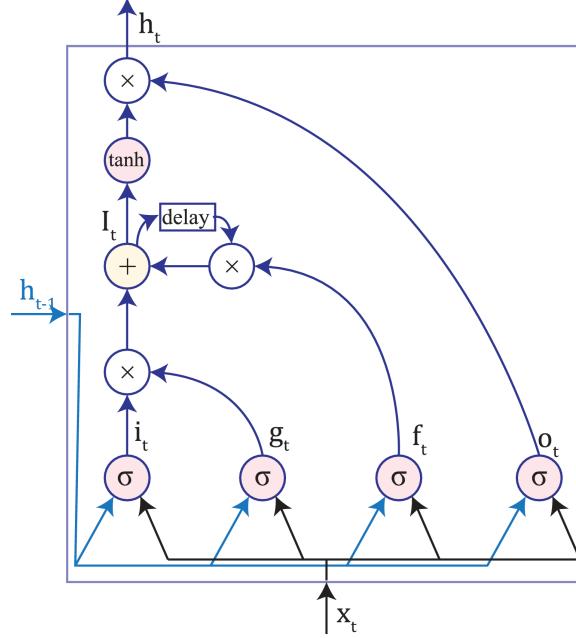


Figure 4.2: LSTM cell structure with an internal gates (Taken from [72]).

reducing the possibility of the gradient becoming worse.

4.3 Natural Language Processing

Natural Language Processing is an old and vast concept that deals with the understanding of language based on the text or speech. Since this thesis deals with the text, we here explore its sophisticated variants and the state-of-the-art models.

4.3.1 Language Modeling

The original application of Language models [85] was to solve speech recognition problems. In fact, they still play a key role in modern speech recognition systems. They are also widely used in other NLP applications. The parameter estimation techniques originally developed for language modeling are useful in many other situations, such as tagging and analyzing problems. Language modeling uses language features and methods through observation to find how words are related, but not by rules, which can become too complicated. It is a probabilistic model that can predict the next word based on the word sequence. In more complex models, a larger sequence-based word context will be considered to complete sentences, paragraphs, or documents. Language models are able to predict the

continuity of sentences as well as generate sentences.

4.3.2 Encoder-Decoder Model

The general architecture of NLP using deep neural networks is the encoder-decoder model [197]. The network is comprised of two parts: encoder and decoder. The encoder converts the input data into an internal representation (lower dimensional than the original input). The decoder uses this internal representation to perform tasks designed for the network. Encoders are usually not task-specific and can be pre-trained for other tasks.

A common method for training non-task encoders is to create a codec model whose task is to recreate the input from its hidden representation. This model is referred to as an autoencoder. The key is that the internal representation contains enough information to accurately reproduce the input, so it can be effectively replaced. Because it has fewer functions than the input, the network working on the representation learned by the encoder requires fewer parameters and can train faster or converge in fewer training steps. This model has the added advantage of enabling multiple decoder training with the same encoder and vice versa. This is a method of machine translation where an encoder is trained for each source language and a decoder is trained for each target language, rather than training a complete network for each source-target pair.

4.3.3 NLP Tasks

From part-of-speech tagging (POS) to the dialogue system, NLP has been developed to now provide a vast range of tasks to solve problems of all scales. Generally, complex NLP tasks are divided into several sub-tasks to achieve the desired goals. Whatever the approach, ML or DL, NLP tasks can be listed in the following categories:

Text Classification Tasks

Text classification usually does not have to maintain word order. This task usually uses a method similar to a set of words to process the entire corpus. It is used to predict labels and categories based on main content, but it is also very common to observe sentiment analysis. It is used to detect offensive language and spam, and it supports proper document classification.

Word Sequence Tasks

Unlike text classification, the order of words is important for this kind of task, because it operates in a word sequence. The order of words is particularly important in language modeling (Section 2.3.1), so the derived task includes predicting the previous word and the next word. Some models can extend predictions to complete sentences. Another common possibility is to generate text recursively derived from predicting the next sentence. The known uses for these kinds of tasks include Named Entity Recognition (NER), POS tagging, language translation, and text completion.

Sequence to Sequence Tasks

This category is built upon word sequencing. These tasks are also called seq2seq, where the input sequence is taken, and the transformed sequence is generated as output. To this end, encoder-decoder model and hidden representations are used. Some typical applications include translations, summarization, Question Answering (QA).

Dialog Systems

NLP is critical to the strength of conversational agents. These systems require high performance in natural language to correctly detect user intent. In addition, it should give appropriate answer. To this end, the system should be capable of combining tasks in the categories listed above to perform applications related to understanding and generating responses. Depending on the scope, knowledge about the world must be integrated.

The dialogue system can be placed into two primary categories: goal-oriented and dialogue-oriented. The first purpose is to realize the user's intent in a specific context and usually replaces the graphical user interface in which the desired transaction will be communicated. Many companies integrate goal-oriented dialogue as the interface to services, a clear application in the hospitality industry, where reservation services are provided. A pure conversational agent has no other purpose, only to keep the dialogue as human as possible. They posed challenges beyond NLP, and produced countermeasures that included maintaining a state of dialogue, logical reasons for input through global knowledge, or full attention to the topics discussed. In other words, the agent needs certain memory and active learning ability to imitate human dialogue.

4.4 Transfer Learning

Transfer learning defines a method in which the basic model of a given source domain of task A can be reused to solve another target task B that may belong to another target domain. The main idea is not to create a complete model for each specific task from scratch, but to further create a basic model with additional data more suitable for the target task. The categories of transfer learning are classified according to different situations. First, if the data is selected in the original domain and the target domain. Second, the difference between the original task and the target task. Figure 4.3 summarizes the existing taxonomy.

When learning is successfully transferred, that is, when the target task is completed using a customized model, a scheme is called positive forward transfer, as proposed by Ruder [177]. On the contrary, in the opposite case, when the adjustment impairs the execution of the target task, a negative forward transfer can be observed. After additional training, the performance degradation of the pre-trained model is usually due to differences in new entries. For example, when the data sets used for pre-training and adaptation are too far apart, such as two completely different languages, the weights of the model may return to a random state and lead to an observation called catastrophic forgetting. The reduced amount of data needed to adapt the basic model to different domains, cost-effective training, and good overall results are the primary advantages that enable DL experts to quickly develop new models to handle various tasks.

4.4.1 Fine-tuning

A well-known method of using transfer learning for neural networks is to copy and create the top n layers, where n is a variable that can be selected according to the particularity required by the target task's behavior. Two methods were created. First, it is a fine-tuning method in which errors in specific tasks will be back-propagated, and the original weights will be reset. Second, the frozen layers method, where it only learns new data in the last layer and the weights of the remaining copied layers will remain unchanged. The rationality of these methods is related to Yosinka's, conclusion about the possibility of transferring features in deep neural networks [225]. Research shows that the first or lower layers of deep neural networks usually encode general information while the last or higher layer becomes increasingly detailed. Through transfer learning, the first layer is considered more

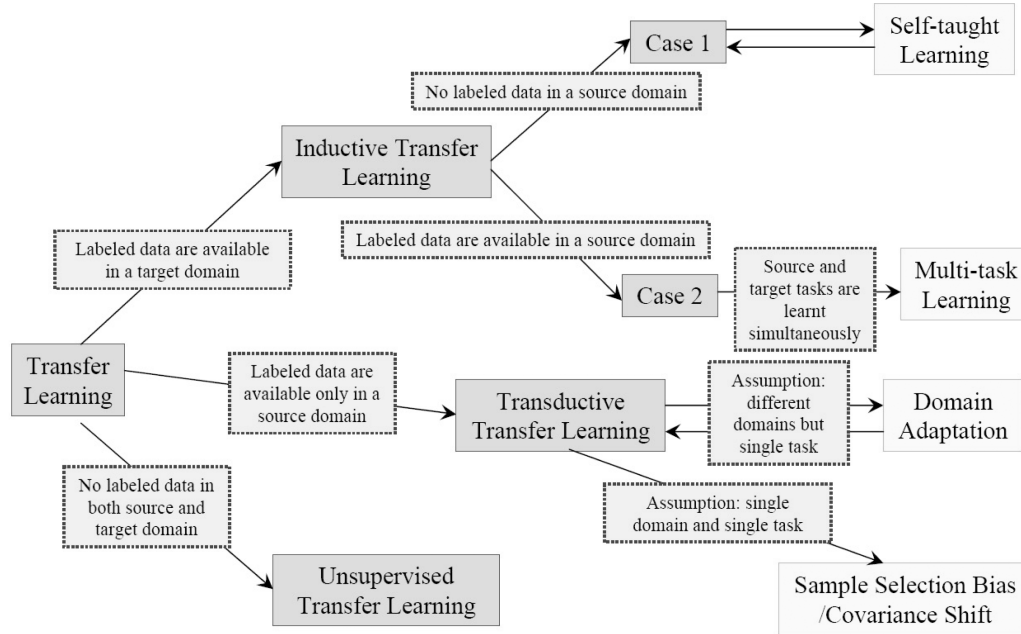


Figure 4.3: An overview of different categories of transfer learning (Taken from [154])

valuable, as it can summarize and support a wider range of domains.

4.4.2 Domain Adaptation

Domain adaptation is a form of transfer learning. The task remains unchanged, but the domain changes, or the distribution between source and target changes. For example, consider a model that has learned to classify reviews of electronic products based on positive and negative sentiments and is used to evaluate hotel room or movie reviews. The task of analyzing emotions remains the same, but the domain (electronic and hotel room) has been changed. Due to the changes between the training and test data (often called domain shift), applying the model to separate domains creates many problems. For example, sentences containing phrases such as "loud and clear" are generally considered positive in terms of electronic products but are considered negative in hotel room evaluations. Similarly, the use of keywords such as "long" or "boring" that may appear in book review fields may not exist at all in fields such as kitchen equipment reviews. Many studies have shown that inserting domain specific vocabulary leads to better performance of the language models [150].

4.5 BERT

In October 2018, Google published a Nobel language model called Bi-direction Encoder Representations from Transformers (BERT) [64]. Its performance shows that it is able to score state-of-the-art results in eleven NLP tasks. The comparison of BERT is shown in [228] that shows it outperforms the state-of-the-art created by Elmo [159] on six distinct non-trivial NLP tasks. The BERT model is composed of 12 bidirectional transformer encoder blocks [209], 768 hidden layers, and 110M parameters and is intensively based on Attention mechanisms [134]. Here, we will introduce details about Attention and Transformers that are important to understand the concept of BERT. We also explain the details on the pre-training and fine-tuning of the model.

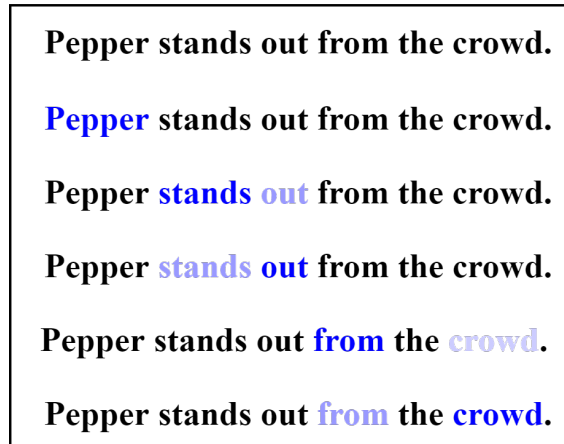
4.5.1 Attention

Neural networks used to form the encoder-decoder models described here are designed for solving sequence to sequence (*seq2seq*) problems [198] using a fixed length context vector for internal representation. This limitation of having a fixed size length vector makes it inefficient to deal with longer sequences, as it forgets the initial inputs. Attention mechanisms, on the other hands, solve this problem through the addition of a layer that normally sits between the encoder and the decoder, focusing on the context vector that helps the decoder to capture the entire global input sequence. This layer does not look into the original input. Rather, it assigns weight to the output of the encoder, then calculates the weighted sum and feeds it to the decoder. It acts as a memory and provides the hidden state of the encoder nodes to decoder.

In a work on machine translation using deep neural networks, Bahdanu [29] explains about an alignment model to train neural network to generate accurate translations with the help of attention. Given two inputs in any two languages, the model predicts score to the best matching words between two inputs. To design this model, author used a bidirectional RNN encoder along with an alignment function that predicts score for each pair of words. Here, we will explain other sub categories of attention that slightly differ in some properties called Self-Attention and Multi-Headed Self-Attention.

Self-Attention: Self-attention, which is also known as intra-attention, is an attention mechanism that creates different positions of a single sequence so that a representation of

the same sequence is computed, as shown in Fig 4.4. Each word in the input sequence is paired with the last words with the highest attention score for computing the related representation. It has been a ground breaking work in machine reading, abstractive summarization, or image description generation.



Pepper stands out from the crowd.
Pepper stands out from the crowd.
Pepper stands out from the crowd.
Pepper stands out from the crowd.
Pepper stands out from the crowd.
Pepper stands out from the crowd.

Figure 4.4: Representation of self-attention mechanism in a sentence.

4.5.2 Transformers

A transformer is a type of neural network architecture proposed by Vaswani et al. [2017]. The transformer consists of the encoder and decoder but completely differs from the structure of RNN and CNN, as shown in Fig 4.5. It relies on multi-head self-attention mechanisms in both encoder and decoder. Such design has been proven to be very effective for solving language-oriented problems, such as language translation and syntactic parsing.

Positional Encoding: The positional encoding is responsible for appending the necessary information about the position of the tokens in the input sequence. This is a very important parameter, because the transformer model relies neither on recurrence nor convolutions. Sinusoidal and cosine functions are needed to encode the position of the token and the vector's dimension.

Multi-Headed Self-Attention: We can now see the process of encoder as input embeddings into the sub-layer of attention referred as multi-headed self-attention. Let us explain the different view adopted by the original paper. It considers the attention function as a mapping query Q and a key-value pair (K, V) where the output is given by the

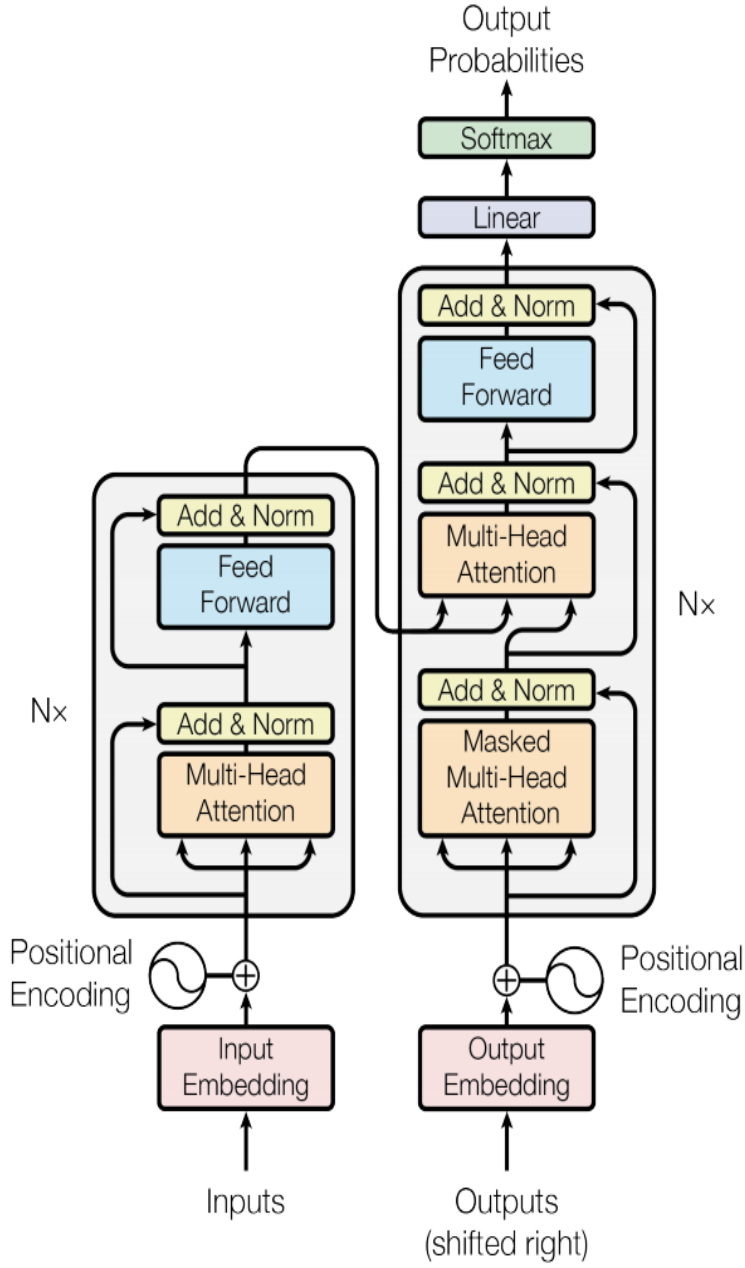


Figure 4.5: Transformer Architecture [209].

weighted sum of the values. The alignment or compatibility function is described as :

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (4.2)$$

where d_k is the dimension of the queries and keys.

A softmax function is a normalization function which accepts a vector z of K real numbers and transforms the components of the vector into probability distribution. This process bounds each component z_i inside 0 and 1 whose total sum gives us 1. The formula is given by:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \text{ for } i = 1, 2, \dots, K \quad (4.3)$$

The multi-headed model is computed through the scaled dot-product attention many times in parallel that results scaled dot-product attentions to be concatenated and linearly transformed into desired dimensions. It is claimed that this process allows the attention to take different subspace representations at different positions into consideration for improving the performance of single head attention mechanism. The process is clearly illustrated in Fig. 4.6. The output of the attention layer is then normalized and passed to the feed forward network. Moreover, the output of the encoder is fed to decoder. Here, the masking and shifting of the output embeddings to the right are the conditions to make sure that predictions are based on the known outputs of the previous positions.

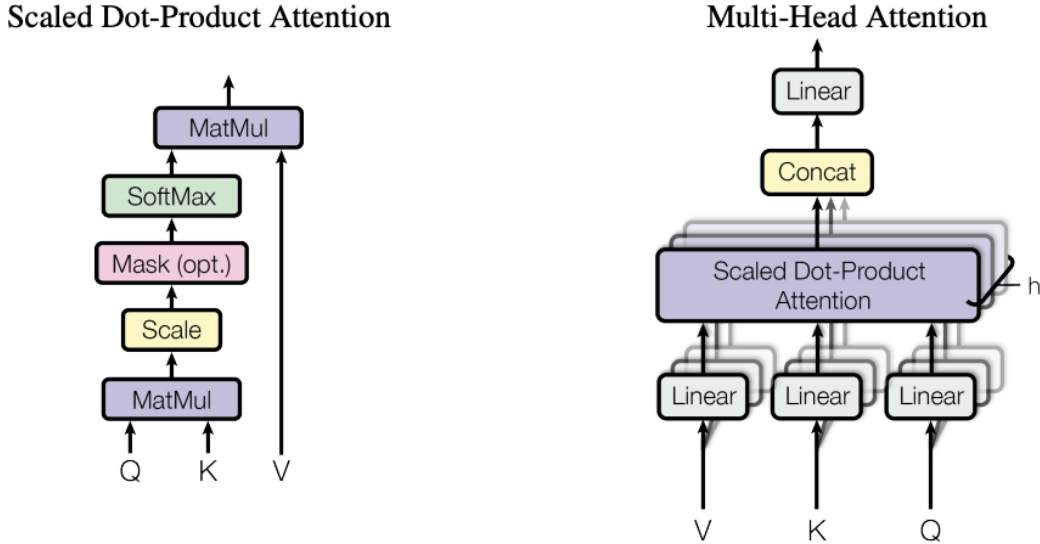


Figure 4.6: Illustration of multi-headed self-attention [209].

Residual Connection and Normalization: From Fig. 4.5, we can see that the residual connections or skip connections that sub-cuts every sub-layer to the normalization layers.

These residual connections support both the forward and backward propagation and are the mechanism that eliminates the problem of vanishing and exploding gradient. Additionally, the normalization is enforced to eliminate the co-variance shift that specifies the distributions of the training and test sets being distinctive.

Feed Forward Networks: This sub-layer is responsible for transforming the output of the encoder and decoder that are fully connected in the feed forward network. In [209], they have implemented two linear transformations with ReLU activation between them.

4.5.3 Model Pre-training

BERT has been pre-trained for 40 epochs on 3.3 billion words of English corpus combining the English Wikipedia dump of 2500M and Book Corpus of 800M. BERT is trained using two techniques: Masked Language model (MLM) and next sentence prediction.

Masked Language Model: MLM strategy is a kind of language modeling that BERT uses that not only considers the sequence preceding the word but also the surrounding context of the given word. BERT heavily relies on bi-directionality that models the mask randomly around 15% of the input tokens. This transforms an unsupervised text corpus into supervised model. In brevity, a setup is designed where a language model predicts only masked tokens, which makes the model learn the representations accurately in both the directions at the same time.

Next Sentence Prediction: Next sentence prediction is another language modelling method which BERT uses to predict the next sentence. This helps BERT to understand the relationship between sentences in given language. This model selects a sentence pairs A and B and in two ways. (i) B is the next sentence to A labelled as TRUE and (ii) B is randomly selected sentence not related to A marked as FALSE. Model learns correlation of the next sentence by training 50% on both the case.

4.5.4 Model Fine-Tuning

The input sequence of the BERT is an array of tokens processed with a special hidden state token [CLS] at the beginning along with a [SEP] separator at the end of the sentence. [CLS] tokens are responsible to encode the whole input for classification tasks. [SEP]

tokens not only acts as the sentence terminator but also show the split between sentences if includes more than two sentences. Figure 4.7 shows the input model for BERT.

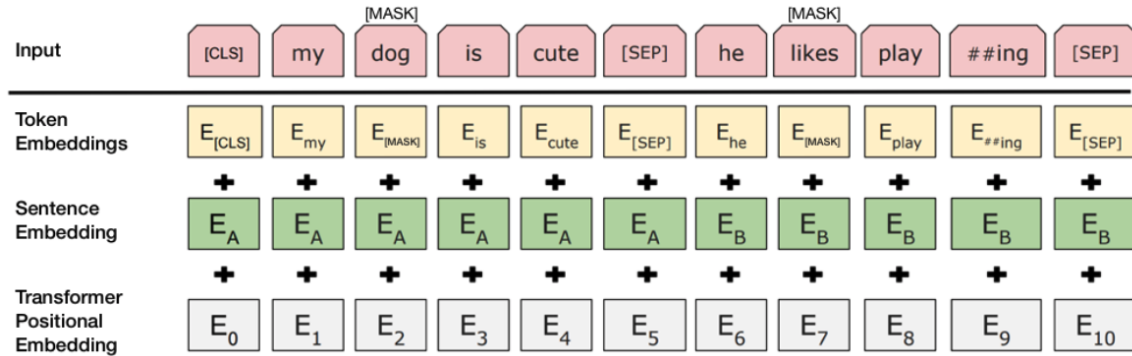


Figure 4.7: Input Illustration for BERT [64]

BERT uses a very sophisticated tokenizer called Word Piece tokenizer that helps to split words into more than two tokens. The main aim of tokenizing words into sub words is to identify more words in the text with limiting vocabulary size. For example, a word “dancing” is sub divided into “danc” and “##ing”. The prefix ## shows that this token is the sub words of another original word. This technique helps in eliminating out of vocabulary problem.

Fine tuning is usually done by feeding task specific input following the necessary pre-processing explained earlier. It is very necessary for designing the tasks with same pre-processing to fit into the BERT model. The major tasks for fine tuning BERT are sentence pair classification, single sentence classification, and QA and single sentence tagging as shown in Fig 4.8. However, it can be expand to other tasks if needed. Despite its fascinating model to learn language model, it has a limitation of the fixed sequence length of 512 tokens that includes [CLS] and [SEP] tokens.

4.6 ALBERT

A Lite Bidirectional Encoder Representations from Transformers (ALBERT) [122] is an innovative new natural language processing (NLP) algorithm. Accuracy score comparisons such as the one shown in Figure 1 establish it as a superior algorithm compared with other models. Additionally, ALBERT requires a fewer number of parameters compared with its larger, more cumbersome previous version, Bidirectional Encoder Representations from Transformers (BERT). In real-world applications, however, practitioners are unlikely

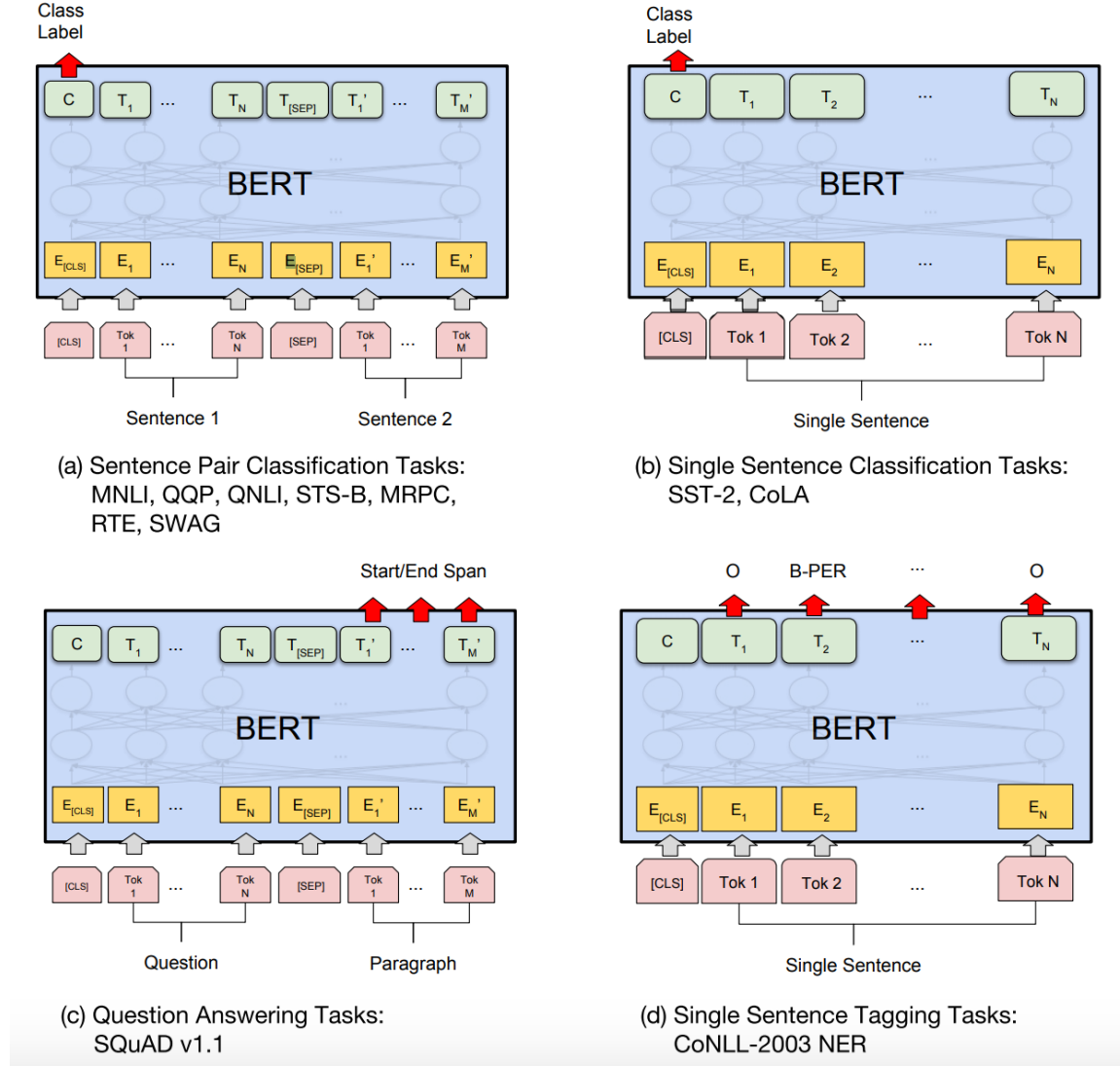


Figure 4.8: Downstream tasks fine-tuning using BERT [64]

to find the smallest version of ALBERT any faster or more accurate than the original base version of BERT. The benefits of ALBERT instead lie in the significantly improved capacity compared with the largest BERT variant, including higher NLP benchmark scores achieved through successful scaling at these larger model sizes 4.9. As a result, ALBERT is capable of fascinating innovation.

The BERT architecture has two dimensions that can be scaled up: (1) depth, or the number of layers and (2) width, or the hidden size (i.e. the number of features in the embeddings output in each layer, or Transformer Encoder, of the model). As shown in 4.10, the BERT architecture peaks at BERT-large, which has a hidden size of 1024. Specifically, increasing the hidden size from 1024 to 2048 significantly reduced the benchmark accuracy,

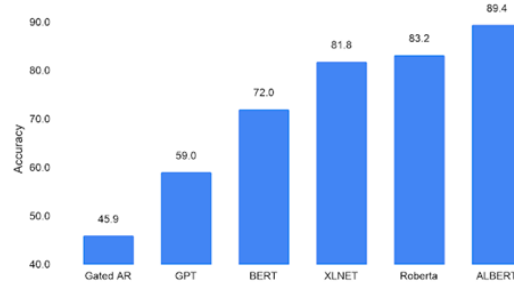


Figure 4.9: Increasing hidden size of BERT- large leads to worse performance on RACE [1]

indicating that the peak performance for the BERT model is achieved at lower hidden sizes, and increases to the hidden size only negatively impact the results. Meanwhile, the highest reported benchmark scores for the ALBERT model, which notably exceed those produced by previous model versions RoBERTa and XLNet, are produced using their ALBERT-xxlarge variant. This model has 12 layers and a hidden size of 4096. This is half the depth but four times the width of BERT-large, demonstrating successful scaling capability, unlike the scaling performance observed for BERT-large. There are a total of six primary changes that the authors made to BERT to create the ALBERT model. These six changes will be summarized in the next section, then assessed in detail in subsequent sections

| Model | Hidden Size | Parameters | RACE (Accuracy) |
|----------------------------------|-------------|------------|-----------------|
| BERT-large (Devlin et al., 2019) | 1024 | 334M | 72.0% |
| BERT-large (ours) | 1024 | 334M | 73.9% |
| BERT-xlarge (ours) | 2048 | 1270M | 54.3% |

Figure 4.10: Increasing hidden size of BERT- large leads to worse performance on RACE [122]

Fig 4.11 lists all model configurations considered by the authors is [122] . The highest benchmark scores are achieved by the ALBERT-xxlarge variant, which uses only 12 layers but length 4096 embeddings. In 4.11, ‘Parameters’ are listed in terms of the number of unique parameters, which is helpful for assessing the model’s memory requirements. ALBERT’s memory requirements are dramatically smaller compared with BERT. The ALBERT-xxlarge model uses only 235M parameters compared with the 1270M used by the BERT-xlarge model. The compute requirements of each model, however, are a different story. Examining the ALBERT-xxlarge variant, for instance, the model would have approximately $231M \times 12 = 2.77B$ weights*, which is twice as large as ‘xlarge’ BERT

With regards to regularization in reference it is suggest that reducing the number of

| | Model | Parameters | Layers | Hidden | Embedding | Parameter-sharing |
|--------|---------|------------|--------|--------|-----------|-------------------|
| BERT | base | 108M | 12 | 768 | 768 | False |
| | large | 334M | 24 | 1024 | 1024 | False |
| | xlarge | 1270M | 24 | 2048 | 2048 | False |
| ALBERT | base | 12M | 12 | 768 | 128 | True |
| | large | 18M | 24 | 1024 | 128 | True |
| | xlarge | 60M | 24 | 2048 | 128 | True |
| | xxlarge | 235M | 12 | 4096 | 128 | True |

Figure 4.11: The configurations of the main BERT and Albert analyzed [122]

parameters in BERT, is achieved using a combination of parameter sharing and embedding size reduction (discussed in the next section), act as a form of regularization. The concepts of ‘regularization’ and ‘generalization’ relate to the common issue in machine learning of ‘over-fitting’. Over-fitting occurs when a model is allowed to fit too tightly to its training data, resulting in highly accurate predictions for data the model has seen but an inability to “generalize” to new, unseen inputs. Over-fitting is often related to overlap between the classes (illustrated by the overlap between the blue and fuchsia classes). This overlap forces the model to create a very complex decision boundary to try and separate the data points. This may work well for the irregular pattern formed by the data points in the drawing, but it is highly unlikely that all the intricate twists and turns of the overfitted model will align with an entirely new set of points from the two classes.

A common solution to this problem is to intentionally make the model less accurate. This is called ‘regularization’. Regularization produces a simpler and smoother decision boundary that is significantly more likely to perform better on new data points compared with the complex decision boundary from the overfitted model. This is the challenge with large neural networks—they have more expressive power, but this does not necessarily produce the most reliable model.

In contrast to BERT and other language models trained on the Adam optimizer, ALBERT is trained using a relatively new optimization algorithm, Layer-wise Adaptive Moments (LAMB) [226]. LAMB is designed to handle training with very large minibatch sizes (i.e. 32,868) in order to speed up training. LAMB’s increased batch size has the important benefit of scaling with hardware very efficiently, so unlike Adam and Adam variants, doubling or quadrupling the hardware can result in a roughly proportional speedup. This is one of the more notable achievements of BERT, where the BERT model was reported to be trained in just 76 minutes by scaling up hardware. This is an impressive speedup

and allows research teams to iterate on language models much faster. When training a network there are some inherent trade offs between training speed and performance. A lot of variables are considered (e.g. training time, batch size, learning rate, optimizer details, generalization, GPU utilization, regularization, etc.) when training, but in discussing LAMB just a few of these will be discussed in-depth, along with the mechanism of how LAMB addresses typical problems faced when training with large batch sizes. In general, larger batch sizes results in better utilized GPUs and lower training time at the expense of model performance. This is because a larger batch size means fewer training iterations per epoch, which needs to be compensated for with a larger learning rate. However, although larger batch sizes have a more accurate estimate of the gradient, a larger learning rate tends to cause more unstable training, leading to poorer performance overall as the model tends to get stuck in a local minima of the gradient. Conversely, small learning rate steps with small batch sizes tends to lead to the best model performance at the tradeoff of training time. LAMB attempts to fix the unstable training update problem of large batches. The authors do this by building off an optimization algorithm called Layer-wise Adaptive Rate Scaling (LARS). LARS is like regular gradient descent, but instead of using a constant or decaying learning rate, the learning rate is adapted per each layer by something called the trust ratio. The trust ratio is simply the Euclidean norm (i.e. the square root of the sum of the squares) of the layer weights over the norm of the gradient update, which at each layer is multiplied with the learning rate to scale it up or down. Essentially, the trust ratio prevents too large of a weight update step in the wrong direction or too small of a step in the right direction at each layer, depending on the gradient.

Question answering (QA) technologies have emerged as effective tools for automated answering of natural language questions posed by humans using either a pre-structured database or list of natural language documents. Put another way, QA systems allow users to ask questions and receive immediate answers using natural language queries, and they can be thought of as an advanced method of information retrieval (IR). Recent development in pre-trained information from huge, unlabelled data has shown promising results in various downstream tasks. Even when initiated with the same pre-trained parameters, each downstream task has its own unique fine-tuned model.

4.7 Acoustics Model Advancements

Natural language is an extremely complex communication form that has become the central focus for many large-scale technological innovations such as a seamless conversion of speech into the text. One of the power tool is Automatic speech recognition (ASR) that directly transcribes recorded speech into text. ASR has exploded over the past decade due to the success of voice activated devices such as Amazon Echo or Siri. Part of the strong performance of these personal assistants is based on their ability to transcribe speech even in challenging conditions such as presence of background noise or unusual speech pattern (i.e. hesitations). These systems typically rely on free speech input once an established keyword is audibly spoken. The complexity of this process requires the use of a cloud, however, there is significant interest in eliminated cloud usage through the development of embedded systems [94, 125, 132, 164, 220]. Current examples of embedded ASR devices include wearable devices and Internet of Things applications. The primary challenges of embedded devices are the hard computing power and memory constraints. To meet these challenges, embedded systems must provide real-time processing with low energy and memory consumption while still meeting the high accuracy of cloud-based neural network systems. Several optimization techniques may be applied to achieve this which include (1) architecture optimization and (2) data quantization and format optimization.

In architecture optimization, the layer quantity and layer sizes are reduced. This process reduces the neural network size with fewer total parameters to achieve a system that can fit within memory-constrained systems. This optimization process also improves the real-time factor. Moving on to data quantization, the weight and activation bit precision are reduced. Data format optimization, meanwhile, focuses on the replacement of floating-point values with fixed-point or integer numbers [140]. Thus, data quantization and format optimization are focused on simplifying operation processes to improve the real-time factor. Since this is an orthogonal optimization, data quantization and format optimization can be applied to any selected architecture to reduce model size while improving the real-time factor, although this may compromise the accuracy of the model.

Pipeline ASR systems involve multiple models—the acoustic model (AM), phonetic model (PD), and language model (LM)—all of which are created in the training stage. Meanwhile, end-to-end ASR systems rely on only one network to learn each of these representations. Both types of ASR systems require large volumes of annotated speech data and

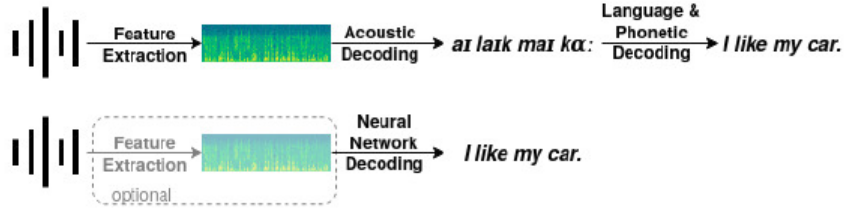


Figure 4.12: Pipeline (top) compared with end-to-end (bottom) ASR models. For pipeline ASR, feature extraction is required, and different output representations are produced during decoding. For end-to-end ASR, raw waveforms are directly transformed into text. Rescoring can be completed in both cases using an added LM. [81]

text.

Pipeline ASR systems are considered the traditional system where multiple models work together in a pipeline format (Fig. 4.12). ASR is completed by finding the most likely phrase, W^* , using the speech signal probability generated by the chosen sequence, as shown in Eq. 1 [81]:

$$W^* = \operatorname{argmax}_p(W|X) \quad (4.4)$$

where X is the speech signal and W is the sequence of words. Using Bayes' rule, this equation can be transformed into the following equivalent form shown in Eq. 2 [81]:

$$\begin{aligned} W^* = \operatorname{argmax}_p(W|X) &= \operatorname{argmax}_p \frac{P(X|W)P(W)}{p(X)} \\ &= \operatorname{argmax}_p P(X|W)P(W) \end{aligned} \quad (4.5)$$

Note that $P(X)$ is independent of the chosen word sequence W . This works to eliminate the denominator, limiting the equation to the following components: (1) probability computation of X given the corresponding word sequence W : $P(X|W)$ and (2) probability computation of the chosen word sequences, $P(W)$. An acoustic model is capable of determining the speech signal probability while the calculation of the word sequence probability requires a LM. A phonetic model is employed to link these two models, as it is composed of a dictionary that uses a phoneme sequence to define each word within the LM, and these sequences are then modeled using the acoustic model.

Shifting back to end-to-end systems, there are several possible interpretations of this type of model (Fig. 4.12). Most often, end-to-end type systems are composed of a sin-

gle neural network where the end-to-end based speech recognition system receives some form of raw audio as the input in order to produce a word sequence output representing the appropriate form of written transcript [213]. Each of the phonetic, acoustic, and language components are all trained within an individual network [187], so a hand-crafted pronunciation dictionary is no longer required. Some end-to-end networks can process raw audio signal [52] where feature extraction is integrated into the network [24]. Hand-crafted features can be extracted within the previously mentioned step. Training of end-to-end type systems is completed using graphemes (i.e. characters) [179], word-pieces [47], or entire words [51]. End-to-end systems provide more output capability over traditional systems (i.e. direct character output vs probability distributions over phonetic units). End-to-end models also offer flexibility in the LM where the LM can either be used as an external add-on or embedded within the network itself. End-to-end networks also offer lexicon-free approaches, offering out-of vocabulary word handling capability, although this feature can increase the risk of meaningless or misspelled word output. End-to-end systems do not use pre-aligned data, as training is performed from scratch to eliminate the occurrence of incorrect alignments for training targets [87]. An additional generative component (i.e. Hidden Markov Model (HMM)) can also be introduced in the form of a hybrid HMM-DNN approach based on a lattice free MMI (LF-MMI) objective function [92] where initial forced-alignments are not required to initiate training.

Speech signals are not stationary, as its statistics are dependent on the temporal dimension. As a result, analysis is performed over small time frames to account for the naturally occurring quasi-stationary nature of the speech signal. Framing represents the first operation, which consists of signal splitting over short frames (e.g. 25 ms) with some amount of overlap (e.g. 10 ms). The second operation, windowing, is performed during framing. Signal convolution is completed via a Hamming [93] or Hanning [152] filter to smooth border frame discontinuities to avoid frequency artifacts. The smoothing process makes overlapping essential since it helps to retrieve information that may have been lost at the frame border. The third pipeline operation is the application of a Fast Fourier Transform (FFT) to convert the signal from a time to frequency domain, as shown in Eq. 3 [81]:

$$S(n) = E(n) * h(n) \xrightarrow{\text{FFT}} S(w) = E(w)H(w) \quad (4.6)$$

where the speech signal passes through a time domain convolution between the base signal, shown by the pause that occurs during breathing between sentences, and the vocal tract time

response. The convolution in a given time domain then becomes a multiplication operation within a given frequency domain. Eq. 4 [81] shows the means by which the logarithm is used to transform the multiplication operation to a linear summation operation:

$$\begin{aligned} S(w) &= E(w) * H(w) \xrightarrow{\log} \log(S(w)) \\ &= \log(E(w)) + \log(H(w)) \end{aligned} \quad (4.7)$$

As shown in Fig. 4.13, though the initial operations are common for most feature types, fewer or additional features can be processed depending at which stage they are created. The spectrograms represent a common feature type defined as the frequency bin power as a set time via the application of framing, windowing, and FFT operations.

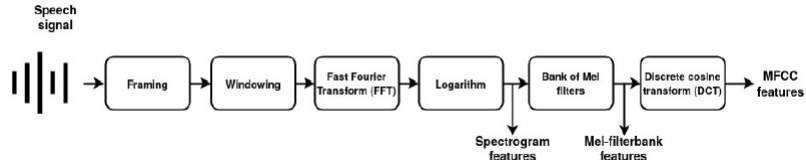


Figure 4.13: Various speech features obtained for each processing stage. [81]

Mel-filterbanks features, also known as Mel-Frequency Spectral Coefficients (MFSC), use essentially the same steps employed by spectrograms. They contain an added step which involves the application of a triangular Mel filter bank [192]. Typically, 40 filters are used which translate frequencies to a narrower range. Vocal parameters are centered around the lower frequency ranges, making this lower spectrum section especially crucial. For the 0-1000 Hz frequency range, the perception is linear and becomes logarithmic for higher frequencies.

MFCCs [61] are the most frequently used speech recognition features obtained using an additional step on top of Mel-filterbanks where a discrete cosine transform (DCT) is applied in the backward transition of the time domain. This step reduces the parameter dimensionality to achieve their decorrelation. Typically, the first 13 MFCCs are retained from each frame signal, although more values provide improved accuracy at the expense of computational complexity.

I-vectors (identity vectors) [62] speaker recognition task-related features, although they also apply to speech recognition. I-vectors are derived using Joint Factor Analysis [115] (JFA) and are comprised of the average (mean) components in the Gaussian Mixture Model

(GMM), which models speaker-specific acoustic features.

Phonemes are the shortest sound units, usually corresponding to the pronunciation of a single alphabet letter. Since phoneme are highly context-dependent, they are typically modeled with three states—(1) shift from the previously occurring phoneme, (2) the stationary, central portion of the current phoneme, and (3) shift to the upcoming phoneme. This form of modeling is known as triphoneme [133] with each state being referred to as a senone. Typically, vectors representing senones are modeled using GMMs [232], and transitions between senones (i.e. triphonemes) are modeled using HMMs [227]. HMM-GMM systems are trained by the Baum-Welch algorithm [32] while speech decoding is completed using the Viterbi algorithm [211].

Speech modeling can be accomplished using HMMs, a finite state automata where the state sequence is unknown and acoustic vectors that correspond to individual states are generated using the probability density function. GMMs are typically trained using an Expectation-Maximization algorithm where the class-belonging probabilities of initial input data are computed, then model parameters are computed using the current class-belonging input data probabilities. The algorithm is allowed to iterate until a convergence at the local maximum for the plausibility function is achieved.

To apply DL to speech recognition, GMMs were replaced with DNNs to model senones by predicting the senone class [101]. As such, hybrid HMM-DNN models are used to predict senone classes using an input of several frames of feature vectors. A Time-Delay Neural Network (TDNN) [212] is a convolutional network that operates within a given time domain to model temporal dependencies. TDNNs are more easily parallelized compared with recurrent networks, making them comparable to feed-forward DNNs with respect to the required amount of training time. In TDNNs, the lower layers are taught a narrow context, and activations are processed in higher layers to widen the temporal context. An optional subsampling technique can be used within the TDNN [158] where hidden activations are computed only at specified (not all) time steps. This mechanism is comparable with a convolutional operation and also reduces the model size and training time while allowing gaps within the convolutional filter.

A factored form of TDNN (TDNN-F) is suitable for network layer compression. A TDNN-F originates from a Single Value Decomposition (SVD) and was introduced as a superior version of the original TDNN [162] since it assumes training from a random start-

ing point and learned matrices are decomposed into the product of two shorter factors, one of which are constrained as semi-orthogonal. This process is achieved using a linear bottleneck operation where the semiorthogonal matrix, M is defined as $MM^T = I$, or $M^T M = I$.

An alternative variation of the TDNN approach adds stacked sets of CNN layers (CNN-TDDs) before implementing the TDNN to complete temporal convolutions of speech features, thus reducing the variability of spectral and temporal aspects. These CNN layers use local connectivity structure, weight sharing, and pooling to remove small variations within the spectral domain caused by the speaker and surrounding acoustic environment [20]. Some literature studies [16, 38, 116, 234] report improved results using CNN-TDNN compared with TDNN networks at the expense of higher computing power requirements.

Encoder-decoder architecture [28, 44, 196] is a neural network specializing in sequence-to-sequence mapping. Typically, recurrent networks are initialized using the encoder and decoder and are jointly trained. A data sequence is inputted through the encoder to produce a fixed-length context vector. The decoder is auto-regressive, as it consumes previously decoded symbols in addition to the context vector to predict the next symbol.

The attention approach is an improvement of the encoder-decoder architecture since it bypasses the limitations of a fixed-length encoding vector. Reports in the literature show that the encoder-decoder network performance decreases with increasing input length [48]. Using the attention approach, the input is encoded to form a sequence of different vectors, a subset of which are chosen during decoding. The decoder will process aligning and decoding operations simultaneously. When the network moves to produce new output symbols, a group of locations within the original source sequence will be examined where relevant information is most concentrated. As a result, a context vector is produced for each of the output symbols. One variation of an attention-based encoder-decoder uses a completely convolutional encoder composed of time-depth separable (TDS) blocks [95]. The TDS block is composed to two parts—a 2D convolution layer and two convolutional layers. This approach provides the advantage of improved generalization while reducing the quantity of parameters and also maintaining a large-size receptive field.

A typical DNN acoustic model relies on frame-level objective functions for training using HMM-GMM system-produced alignments. The cross-entropy function (CE) is commonly used to provide these alignments and is described in Equation 5 [81]:

$$F_{CE}(\lambda) = \sum_{t=1}^T \sum_{i=1}^I \hat{y}_i(t) \log y_i(t) \quad (4.8)$$

where t is iterated over the training set i for set time frame, and \hat{y}_i corresponds to each frame provided using forced-alignment.

Connectionist Temporal Classification (CTC) [86] facilitates network training without the requirement of frame-level alignment for the speech signal transcript and training dataset transcript. A softmax component is used as the final network layer to determine the probability distribution for all potential output symbols. The output is described by a $G_{CTC}(\theta; T)$ transcription graph θ for T time frames. Each node is represented as a possible output label given by the probability distribution function, $F_t()$. The path $\pi = \pi_1, \dots, \pi_n \in G_{CTC}(\theta; T)$ is the possible transcription throughout the graph. The following sequence level objective function is derived from the maximum likelihood and maximizes the likelihood of a correct symbol [81]:

$$CTC(\theta, T) = -\logadd \sum_{t=1}^T f_{\pi_t}(x) \quad (4.9)$$

where $\logadd()$ is an exponential function applied to a logarithmic summation and is an improved upon version of the original $\max()$. An alternative version of the CTC approach incorporates a loss function to join the CTC and attention approach [103].

The Lattice-Free Maximum Mutual Information (LF-MMI) objective function is utilized for chain models [163] for sequence-discriminative training. A traditional MMI maximizes posterior probability using the following relationship shown in Eq. 7 [81]:

$$\begin{aligned} F_{MMI}(\lambda) &= \sum_{r=1}^R \log P_{\lambda}(S_r | O_r) \\ &= \sum_{r=1}^R \log \frac{P_{\lambda}(O_r | S_r)^k}{\sum (O_r | S)^k P(S)^k} \end{aligned} \quad (4.10)$$

where S_r represents the accurate transcription for the r^{th} speech file O_r and $P(s)$ is the

LM probability for a sentence s . In Eq. 7 the numerator is the probability of data giving the accurate word sequence, also known as reference alignment, and the denominator is the total data probability, provided with the complete list of potential word sequences, equaling the sum of all potential word sequences as estimated using the entirety of both the AMs and the LMs. Additionally, the numerator encodes supervision information, specific for each occurrence while the denominator encodes each potential word sequence, which are identical for each occurrence.

Prealigned data is not used to train chain models. Although previous attempts to use CTC to outperform cross-entropy were reportedly unsuccessful, it has been suggested that some CTC concepts could be applied to the sequence-discriminative LF-MMI criterion [163]. Although both CTC and MMI maximize the conditional log-likelihood of a correct transcript, the probability in CTC is locally normalized as opposed to the global normalization of MMI.

Auto Segmentation Criterion (ASG) [9] is a developed improvement to CTC [52] that introduces dependency between output symbols. This is accomplished using transitional probabilities between the output symbols. Additional advantages of ASG include a less complex output graph and un-normalization of node scores, which facilitates the external LM plug-in. The global normalization of ASG results in low confidence for incorrect transcriptions. Specifically, the score for a provided word sequence W is shown in Eq. 8 [81]:

$$ASG(W) = \logadd \sum_{t=1}^T f_{\pi_t}^t + g_{\pi_{t-1}, \pi_t} \quad (4.11)$$

where $f(\cdot)$ is the probability of a particular output symbol at time step t , and $g(\cdot)$ is the transition probability connecting two sequential symbols.

The LM estimates the likelihood of the word sequence $W = w_1, \dots, w_n$ to form a correct sentence. LMs are useful in the decision-making process when the set of phonemes from the acoustic model output could form multiple sentence variations. Several LMs exist. N-gram models use statistical views towards the concept of how words can be combined together to form realistic sentences. This model assumes that words only depend on a set number of previously used words, and the likelihood of a word sequence forms a probabil-

ity set. The probability for any given word is dependent on the proceeding words, as shown in Eq. 9 [81]:

$$\begin{aligned} p(w) &= p(w_1, w_2, \dots, w_n) \\ &= p(w_1) * p(w_2|w_1) * p(w_n|w_1, w_2, \dots, w_{n-1}) \end{aligned} \quad (4.12)$$

The probability of the occurrence or succession of a word can be computed using large volumes of text. The most commonly used n-gram models include 2- and 3-gram, which require a one- or two-word history, respectively. The probability for a word pair using a 2-gram model can be shown using Eq. 10 [81]:

$$p(w_j|w_i) = \frac{\text{count}(w_i, w_j)}{\sum \text{count}(w_i, w)} \quad (4.13)$$

where the probability of an occurrence of the word pair (w_i, w_j) is provided by the quantity of occurrences for the word w_i proceeding the word w_j . This is divided by the total counts of occurrences for the specific word w_i followed by another word.

RNNs, meanwhile, are capable of learning from all previously used words. These can be as simple as a single input layer, hidden layer, and output layer [141]. RNNs can be problematic to train using a backpropagation technique as a result of vanishing gradients employed within the model. LSTMs provide a potential solution to this by using an alternative memory cell. Literature reports present a LSTM-based method consisting of a single input layer and two separate hidden layers (one projection layer and one recurrent layer) using the LSTM cells [195].

Another new convolutional network type relies on gated linear units (GLU) and outperforms LSTMs for LM in both accuracy and implementation since it is more easily parallelized with less complexity [60]. In this convolutional network, a convolution operation is performed over the input to remove temporal dependencies.

The Transformer-XL [56] network learns dependencies without fixed-length context constraints. This network is capable of capturing longer dependencies compared with simple Transformers or RNNs and achieves improved performance for both long and short

sequences with a faster inference time.

For shallow fusion [90, 113], the AM attempts one potential phone set for each time step scored using the weighted score sum provided using the AM and LM. The point at which the shallow fusion relationship occurs is referred to as an inference time and is shown in Eq. 11 [81]:

$$\hat{y} = \operatorname{argmaxlog}(y|x) + \lambda_{PLM}(y) \quad (4.14)$$

where the initial term represents the AM probability, and the following term is the LM probability.

Deep fusion [90] relies on concatenation of both the AM and LM hidden states adjacent to one another. The models are separately trained, and fusion occurs via gating mechanism. The fact that both the AM and LM are trained separately is a disadvantage, especially for encoder-decoder models. Also, this method does not work well if the AMs and LMs are trained using separate domain corpora, where the decoder has a tendency to follow whatever style has been learned by the AM. Cold fusion [191] is a derivation of deep fusion where the end-to-end AM is trained from scratch using a pre-trained LM. Cold fusion relies on separate gates for each individual hidden node in the LM, letting the decoder choose what LM information provides the best fit for a set time step. It has been reported that a cold fusion decoder outperforms end-to-end attention-based systems [191]. Reports show other novel LM integration approaches where a pretrained LM represents the lower decoder layer in an attention-based encoder-decoder system to provide tighter word embeddings to the context [201].

ASR output takes a lattice form, or a graph $G(N, A)$, and N represents nodes while A represents arches. Lattice rescoring [142, 217] is the replacement of existing lattice probabilities with those provided by using a more efficient LM. Lattice rescoring is performed for the n-best defined hypotheses produced after a beam search. Meanwhile, shallow fusion is performed using log-linear interpolation of the AM and LM score for individual beam search time steps.

There are several popular ASR systems, along the most popular of which are Kaldi's pure-TDNN [4], Kaldi's CNN-TDNN [3], DeepSpeech2 implementation from PaddlePad-

dle [10], RETURNN by RWTH [11], Facebook CNN-ASG [13], Facebook TDS-S2S [14], Jasper by Nvidia [8], and QuartzNet by Nvidia [9]. The most commonplace ASR tasks and corpora are shown in Fig. 4.14. Of the ASR frameworks, all provide adapted ASR systems for LibriSpeech, the most popular free English dataset [155], although only three have adapted systems compatible with WSJ [157].

| ASR task | Speech type | Size [h] | # of speakers | Framework | | | | |
|------------------|------------------|----------|---------------|-----------|---|---|---|---|
| | | | | K | P | W | R | N |
| LibriSpeech [68] | read speech | 960 | ~2400 | ✓ | ✓ | ✓ | ✓ | ✓ |
| WSJ [69] | | 80 | 284 | ✓ | | ✓ | ✓ | |
| TED-LIUM2 [70] | TED talks | 207 | 1242 | ✓ | | | | ✓ |
| Switchboard [71] | conversational | 300 | 543 | ✓ | | | | ✓ |
| Fisher [72] | telephone speech | 2742 | ~12400 | ✓ | | | | |

Figure 4.14: Comparison of popular speech data sets for ASR applications. This table compares speech type and data set size in terms of speech hours and speaker number. Availability is shown for popular ASR frameworks, Kaldi (K), PaddlePaddle DeepSpeech (P), Wav2Letter (W), RWTH Returnn (R), and Nvidia (N). [50, 81, 84, 155, 157, 176]

The TDNN Kaldi chain model is a lightweight, multicomponent ASR system using a TDNN applicable to AM and a HMM to perform sequence modeling. This hybrid system is comprised of a TDNN-based AM, both a PM and LM. More complex LMs can be incorporated for rescoring to improve the initial transcription. The feature details are summarized in Figure 4.15. Specifically, the left side of Fig. 4.15 shows the network input features, a network the figure center shows a network overview, and the bottom left shows the output blocks. Note that the literature shows that the network performance is improved if there are two output blocks [17].

Kaldi’s CNN-TDNN is an extension of the Kaldi chain model TDNN that processes input features using 1D convolutional layers. This model can also be implemented within the Kaldi toolkit for LibriSpeech task approaches. Figure 4.15 shows the details of this where the organization and feature extraction procedure can be viewed at the center of the left-hand column and the complete CNN-TDNN network can be viewed in the center column.

DeepSpeech2 implementation from PaddlePaddle is an end-to-end bi-directional RNN using convolutional layers to process speech features. This architecture is designed to fulfill LibriSpeech ASR tasks. Figure 4.16 illustrates the details of this single neural network end-to-end system where audio features are processed to provide word output.

The RETURNN from RWTH model is comprised of an attention-based encoder-decoder

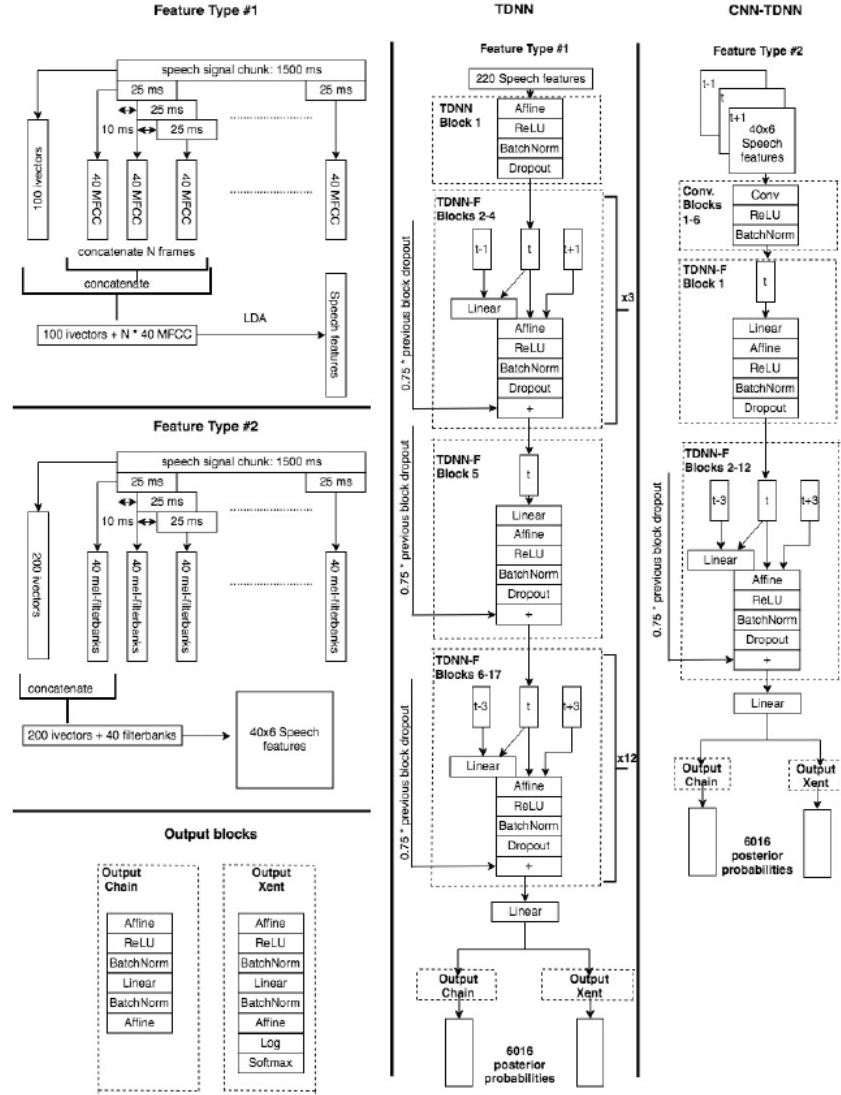


Figure 4.15: Architectures and feature types of Kaldi-based ASR systems. Feature type 1 (upper left) is implemented within TDNN models (center) while type 2 features (middle left) are implemented within CNN-TDNN models (right). Output blocks (bottom left) using cross-entropy functions along with chain loss functions are utilized in both network types. [81]

ASR to produce word part output. This end-to-end system [230] is composed of recurrent layers. The network receives hand-crafted feature input with subwork part outputs created using byte-pair-encoding (BPE) [186]. A detailed summary of this system is shown in the left-hand side of Fig. 4.16.

Facebook CNN-ASG is a fully convolutional end-to-end network utilizing a CTC-based criteria ASG that can output characters. The fully convolutional nature of this system is due

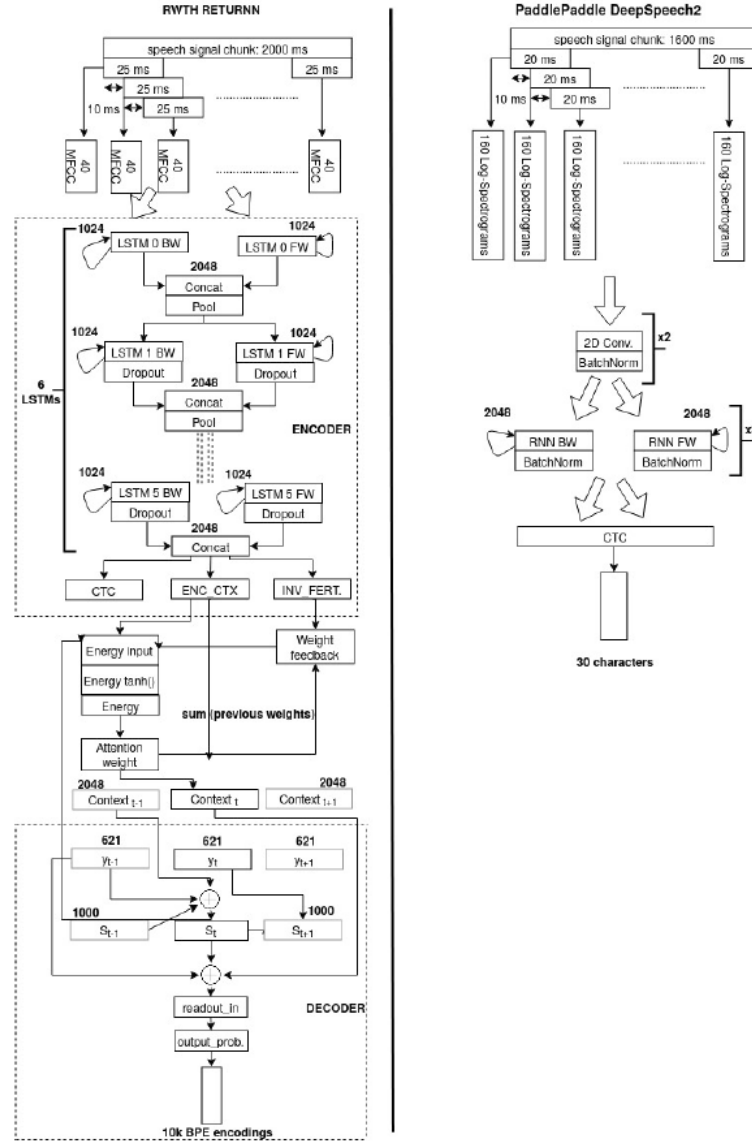


Figure 4.16: RWTH RETURNN model (left) and PaddlePaddle DeepSpeech2 model (right). [81]

to the 17 1D time-convolution blocks that are used, each of which is characterized using a weight normalization operation [180], in addition to the convolution itself and application of the dropout technique. The categorization of this system can depend on several factors. The first is whether the input is raw audio [229], a power spectra, MFCCs, or even Mel-filterbanks. Second is whether the system utilizes a lexicon as opposed to operating lexicon-free [19, 128] where the lexicon behaves as a phonetic model consisting of mapping words as a sequence of acoustic unit tokens. Next is how the system outputs scores over acoustic units (e.g. phonemes, graphemes, word pieces, etc.). The final factor is the system output,

which can be represented by characters or plugged-in via shallow fusion [60]. The system details can be seen in the left-hand side of Fig. 4.17.

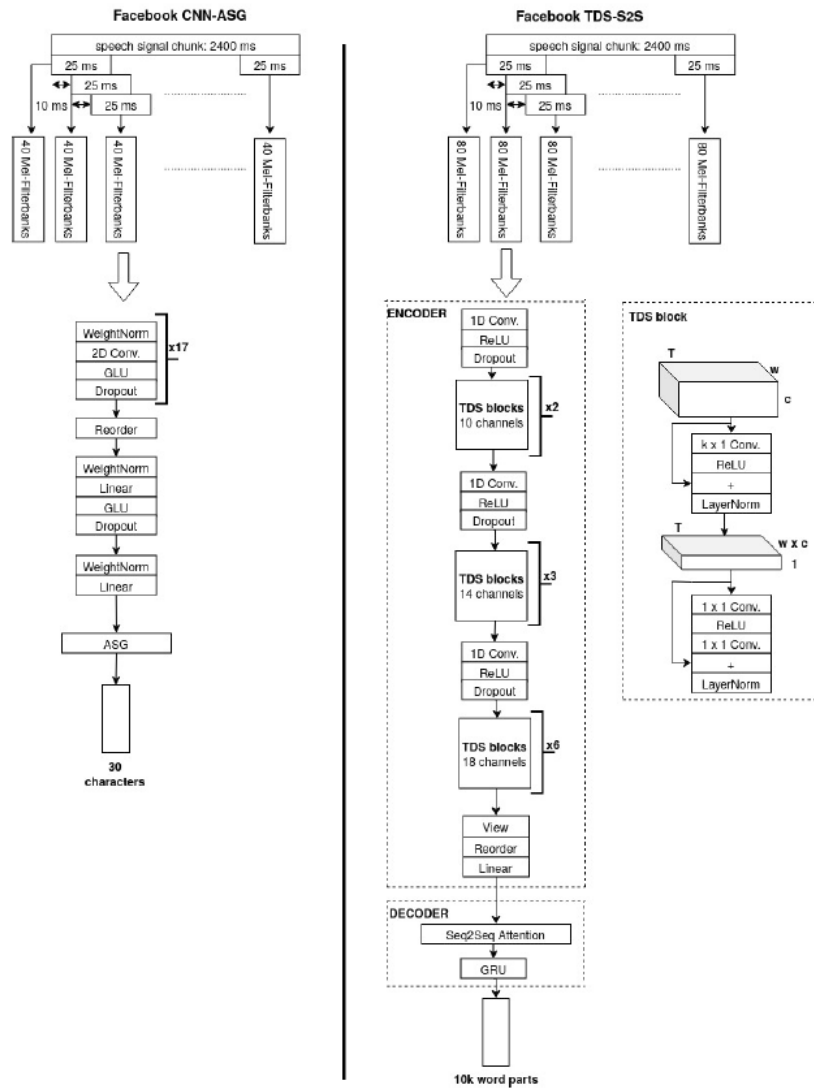


Figure 4.17: Facebook Wav2Letter networks comprised of completely convolutional architecture along with an ASG loss function (left) and encoder-decoder with TDS blocks (right). [81]

The end-to-end deep neural network system offered by Jasper from Nvidia is based on a time-delayed convolutional network interwoven with completely connected layers and is also characterized by residual connections. Jasper from Nvidia is comprised of a single neural network and does not require a phonetic model. As such, the framework can be optionally integrated with probabilistic or even Transformer-XL neural network LMs [56]. A newly developed optimizer, NovoGrad [83], can also be implemented within this network. Although comparable to Adam, NovoGrad is designed to compute second moments

for each layer as opposed to per weight, improving network stability and reducing memory consumption by up to half. The details of this system can be viewed in Figure 4.18 (left).

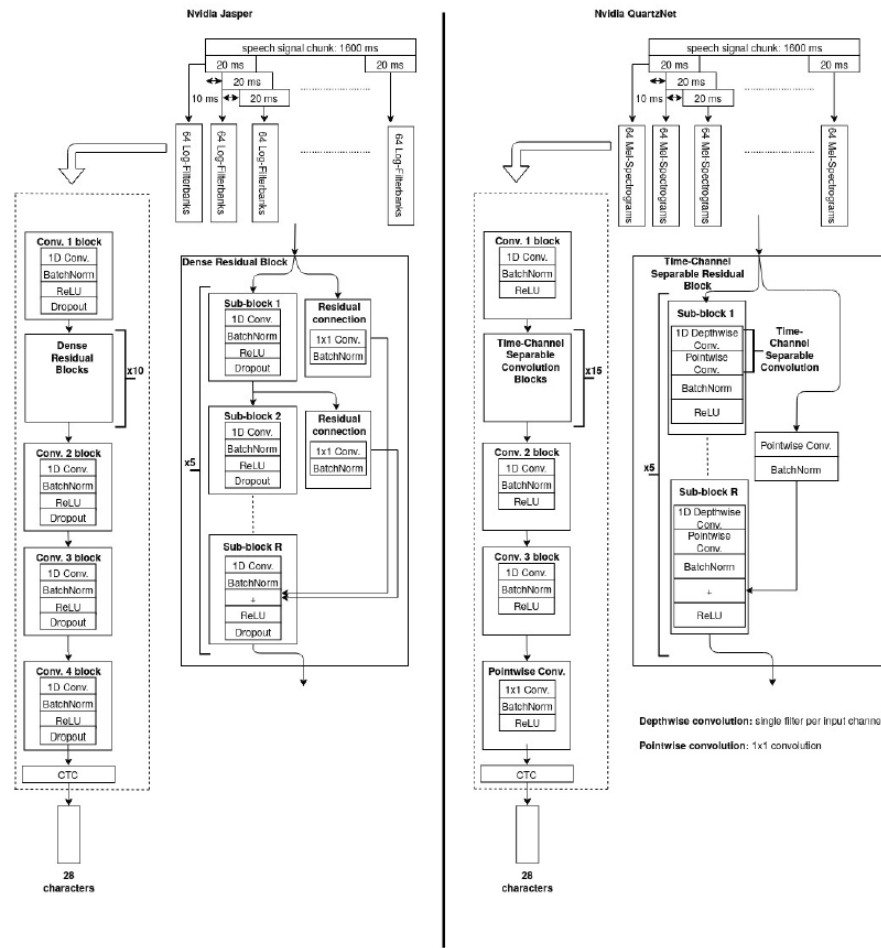


Figure 4.18: Nvidia time convolutional networks: Jasper, which contains dense residual blocks (left), and QuartzNet, which contains time-channel separable residual blocks (right). [81]

QuartzNet from Nvidia [117] is a Jasper-derived end-to-end deep neural network that stems from 1D time-channel separated convolutions. This system is a more efficient version of Jasper regarding the quantity of operations and parameters. Similar to Jasper, the QuartzNet system is also a single neural network using preprocessed feature input with character output along with support for probabilistic or neural network LM integration. The system details are shown in Fig. 4.18 to the right side.

Of the previously mentioned ASR systems, all are end-to-end systems with the exception of the two Kaldi-based systems, which are multi-component systems. For the multi-component systems, the AM, phonetic dictionary, and LM are different components that

| ASR system | Kaldi TDNN | Kaldi CNN-TDNN | PaddlePaddle DeepSpeech2 | Facebook CNN-ASG | Facebook TDS-S2S | RWTH Returnn | Nvidia Jasper | Nvidia QuartzNet |
|---|---------------------------------------|------------------------------------|---|----------------------------|--|-------------------------------|--|---|
| System type | HMM-based | HMM-based | E2E NN | E2E NN | E2E NN | E2E NN | E2E NN | E2E NN |
| Multi-component vs. single NN | AM (NN) + PD + LM | AM (NN) + PD + LM | Single NN | Single NN | Single NN | Single NN + BPE encoding list | Single NN | Single NN |
| | + [Op. LM resc.] + [Op. LM resc.] | + [Op. LM resc.] | + [Op. LM] | + [Op. LM] | + [Op. LM] | + [Op. LM] | + [Op. LM] | + [Op. LM] |
| Speech features | 3 frames x 40 MFCCs + 100 iVectors | 1 frame x 40 fbanks + 200 iVectors | 160 frames x 160 log-spectrograms | 240 frames x 40 Mel-fbanks | 240 frames x 80 Mel-fbanks | 200 frames x 40 MFCC | 160 frames x 64 Log-fbanks | 160 frames x 64 Mel-spectrograms |
| NN architecture | Time delay | Conv. + Time delay | Conv. + Bi-RNN | Conv. + GLU | TDS-GRU Enc. - Dec. | LSTM Enc. - Dec. + Attention | Time delay | Time-channel separable conv. |
| Layers | 1 x TDNN 16 x TDNN-F | 6 x CNN 12 x TDNN-F | 2 x CNN 2 x Bi-RNN | 17 x CNN 1 x GLU | 1 x CNN + 2 x TDS 1 x CNN + 3 x TDS 1 x CNN + 6 x TDS 1 x GRU | 6 x LSTM 1 x Attention | 1 x CNN 10 x Dense residual 3 * CNN | 1 x CNN 15 x TCS Conv. 2 x CNN 1 x 1 conv. |
| Output | AM: 6k posteriors + LM: 200k words | | NN: 30 chars NN: * words + LM: 200k words | | NN: 10k word parts NN: * words + LM: 200k words | | NN: 28 characters NN: * words + LM: 200k words | |
| Loss function | chain + cross-entropy | | CTC | ASG | S2S Attention | S2S Attention + CTC | | CTC |
| Model size [$\times 10^6$] | 20 | 18 | 49 | 208 | 38 | 187 | 333 | 18.8 |
| Operations per frame [$\times 10^6$] | 41 | 63 | 105 | 22k | 15 | 125 | 42k | 1.8k |
| Activations per frame [$\times 10^3$] | 44 | 51 | 13 | 1k | 9 | 38k | 3k | 3.5k |

Figure 4.19: Comparison of ASR Systems. Systems are compared by (i) type—hybrid, HMM-based, and end-to-end type neural network, (ii) component types—multi-component and single neural networks containing additional or optional LM, (iii) speech features, (iv) neural network architecture, which also includes function loss, (v) output type and size, and (vi) model complexity regarding size of the model, quantity of activations, and operation count required to process a single speech frame. Systems are described with respect to complexity, architecture, and hardware requirements. [81]

function as a single system. For the other mentioned systems, only a single neural network is used towards these three components. All mentioned models rely on hand-crafted features for input, although only the Kaldi system uses two components, MFCCs and iVectors, whereas the other uses only one. The Kaldi systems integrate cross-entropy or even chain loss objective functions derived from the LF-MMI cost function whereas the single neural network systems rely on more complex architecture in the form of sequence-to-sequence attention, CTC, and/or ASG. These act as loss functions and HMM to align sequences. The hybrid-approach neural network systems provide posterior probabilities of phonetic units output while other networks provide character or word part outputs. An overview of these characteristics is provided in Figure 4.19.

To determine which models most suitable for embedded system applications, they were used for a variety of operations—(1) source code analysis, (2) inspection of the log files at the inference time, and (3) running the inference for each individual step in debugger mode. This information was then used to determine the complexity corresponding to each individual layer. The formulas used to complete these calculations are summarized in Figure 4.20. Note that the quantity of gates is dependent on the recurrent cell type—1 for RNN, 3 in the case of GRU, and 4 for a LSTM. Also, the fourth component is set equal to either 1

or 2 depending on if the layer displays unidirectionality or bidirectionality.

| Network complexity | Formula |
|---|--|
| Parameters (model size) in fully connected layers | $input\ size * output\ size + bias$ |
| Parameters (model size) in time-delay layers | $(input\ size * output\ size + bias) * context\ size$ |
| Parameters (model size) in convolutional layers | $(filter\ size * number\ of\ input\ filters + bias) * number\ of\ output\ filters$ |
| Parameters (model size) in recurrent layers | $(input\ size + recurrent\ layer\ size) * output\ size * number\ of\ gates * directionality\ factor$ |
| Multiply-accumulate operations (MACs) | $parameters * features\ vector\ length * sequence\ length\ in\ time$ |
| Operations (Ops) | $MACs * 2$ |
| Activations in fully connected layers and time-delay layers | $output\ size * output\ sequence\ length$ |
| Activations in convolutional layers | $output\ size * number\ of\ output\ filters * output\ sequence\ length$ |
| Activations in recurrent layers | $recurrent\ layer\ size * output\ sequence\ length * directionality\ factor$ |

Figure 4.20: Formulas used to determine network complexity. [81]

There is an extremely large variation in the model size from 18M parameters in the case of Kaldi CNN-TNN and Nvidia QuartzNet all the way to 333M parameters for Nvidia Jasper. Of the previously mentioned models, Nvidia Jasper is classified as the most complex with the highest parameter count and quantity of operations required to process a single frame of speech. After Nvidia Jasper comes the recurrent encoder-decoder model from RWTH and convolutional CNN-ASG model by Facebook. Although the RWTH and CNN-ASG models are similar in size (187M and 208M, respectively), the RWTH model requires substantially larger memory for storage of the 38M required activations per frame whereas the CNN-ASG model requires higher processing power for the 22M required operations per frame. Although QuartzNet uses a comparatively small amount of parameters, it uses a medium load with respect to memory due to its substantial activation number. QuartzNet also uses a high amount of operations, although they are lower than Nvidia Jasper and Facebook CNN-ASG. Kaldi’s models are the lightest with only 20M parameters and fastest with 40M-60M operations per speech frame. Facebook TDS-S2S performs similarly to the Kaldi models; the somewhat larger model results in higher memory requirements although the required processing power per speech frame is reduced by 3-4x.

| ASR system | WER[%] | | | |
|-------------------------------|------------|------------|------------|------------|
| | without LM | | n-gram LM | |
| | test-clean | test-other | test-clean | test-other |
| Kaldi TDNN [60] | - | - | 3.85 | 9.57 |
| Kaldi CNN-TDNN [61] | - | - | 3.87 | 9.42 |
| PaddlePaddle DeepSpeech2 [62] | 10.70 | 30.00 | 6.03 | 20.29 |
| RWTH Return [63] | 4.71 | 15.17 | 4.67 | 15.16 |
| Facebook CNN-ASG [64] | - | - | 4.82 | 14.54 |
| Facebook TDS-S2S [65] | 5.36 | 15.64 | 4.21 | 11.87 |
| Nvidia Jasper [66] | 3.86 | 11.93 | 3.19 | 9.03 |
| Nvidia QuartzNet [67] | 3.90 | 11.28 | 2.98 | 8.38 |

Figure 4.21: Comparison of ASR systems regarding performance where performance is measured by rate of word error. This evaluation is performed using two LibriSpeech subsets—test-clean and test-other. For frameworks that allow, two scenarios are used for each evaluation—with and without an external LM. [3, 4, 8–11, 13, 14, 81]

To compare the model performance, the word error rate (WER, [%]) is compared at each implemented system level. The WER is determined as the total count of transcription

errors (i.e. deletions, insertions, and substitutions) regarding the total quantity of words within the groundtruth. The error rates for each model are summarized in Figure 4.21. For more complex models, high-quality transcriptions can be achieved without the use of an additional LM. Results are summarized in Fig. 4.21 for both end-to-end neural networks excluding an additional LM for rescoring (left) and full ASR systems used in conjunction with an external LM (probabilistic non-pruned 4-gram (*fglarge*) [18]) (right). The evaluation was completed using two LibriSpeech test datasets. The first is test-clean, composed of clean speech. The second is test-other, composed of speech recorded using more challenging (e.g. noisier background) sound conditions. The details of these datasets are summarized in 4.22.

Examining the comparison of performance, the complex models (i.e. RWTH RETURNN and Nvidia Jasper) provide similar results despite the use of an external LM. Less complex models with this ability (i.e. Facebook TDS-S2S and PaddlePaddle DeepSpeech2), however, exhibit poor performance in this scenario. Overall, the best results are produced using Nvidia Quartznet, although these are closely followed by Nvidia Jasper and the Kaldi systems. These are followed by the Facebook systems and RWTH RETURNN, and PaddlePaddle DeepSpeech2 produces the worst overall performance.

| Purpose | Set | Size [h] |
|------------|-----------------|----------|
| Training | train-clean-100 | 100 |
| | train-clean-360 | 360 |
| | train-other-500 | 500 |
| Evaluation | test-clean | 5.4 |
| | test-other | 5.1 |

Figure 4.22: Librispeech corpus—various training and evaluation subsets, along with their corresponding size. [81]

The memory load and throughput hardware system requirements of each model are summarized in Figure 4.23. Examining the comparison, RWTH RETURNN, Nvidia Jasper, Nvidia QuartzNet, and Facebook CNN-ASG are all substantial in their memory load requirements (14x-235x the least intensive system, Kaldi CNN-TDNN). Meanwhile, the Kaldi-based systems, PaddlePaddle DeepSpeech2, and Facebook TDS-S2S have similar low memory requirements (100-200MB). Figure 4.24 (left) illustrates the compromise between ASR performance and memory requirements. Thus, while Nvidia QuartzNet needs large memory capacity, it lies within the Pareto front owing to the reduced WER. The Kaldi-based systems also lie in the Pareto front since they dominate either performance (i.e. CNN-TDNN) or memory requirements (i.e. TDNN). Examining the required computational power, Nvidia Jasper and Facebook CNN-ASG require too much power to

| ASR system | Performance | Hardware requirements | | System on Pareto frontier |
|---------------------------|-------------|-----------------------|-------------------|---------------------------|
| | WER[%] | Memory [MB] | Throughput [GOPS] | |
| Kaldi TDNN | 3.85 | 106 | 4.1 | ✓ |
| Kaldi CNN-TDNN | 3.87 | 103 | 6.3 | ✓ |
| Paddle Paddle DeepSpeech2 | 6.03 | 204 | 10.5 | ✗ |
| RWTH Returnn | 4.67 | 23548 | 12.5 | ✗ |
| Facebook CNN-ASG | 4.82 | 1432 | 2200 | ✗ |
| Facebook TDS-S2S | 4.21 | 157 | 1.5 | ✓ |
| Nvidia Jasper | 3.19 | 3132 | 4200 | ✗ |
| Nvidia QuartzNet | 2.98 | 2169 | 180 | ✓ |

Figure 4.23: ASR performance compared with hardware requirement trade-off. Performance is shown as the WER obtained using LibriSpeech test-clean dataset. Hardware requirements are shown with respect to memory load (MB) and minimum throughput (GOPS). Note: regarding the memory load, only the required amount to load the neural mode and store all activations for 1 second of speech processing are considered. More memory may be required for other components. Regarding throughput, only the operations needed to pass speech through the network were considered.

be implemented in embedded systems. The compromise between ASR performance and throughput requirements is shown in Figure 4.24 (right), where Nvidia Quartz Net, Facebook TDS-S2S, and Kaldi TDNN are on the Pareto front. As shown in 4.23, Kaldi TDNN is the best candidate for embedded systems when considering performance and hardware requirements.

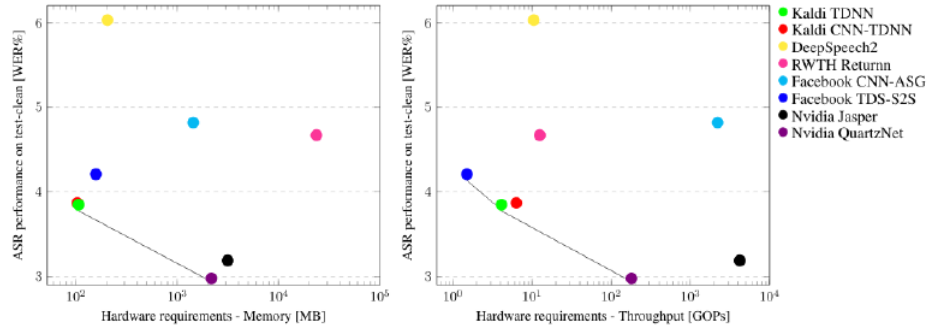


Figure 4.24: ASR performance compromises between memory requirements (left) and throughput requirements (right). [81]

4.8 Introduction to Bayesian Inference

The main aim of the thesis is to evaluate a feasible response for the QA model. In order to achieve that we have used the Bayesian inference which is one of the most important statistical inference recently. So we first explore the basic concept of Bayes' theorem leading to its application in our thesis.

Bayesian statistics is a data analysis approach that uses Bayes' theorem to update available parameter information within a statistical model with information from observed data [205]. Conceptualized in 1763, Bayes' theorem on inverse probability was first introduced in 1825 where the probability of future events were calculated using past event data [205]. In Bayesian statistics, all statistical model parameters, both observed and unobserved, are treated with a joint probability distribution known as the prior and data distributions [205]. Building off this basic concept, Bayesian data analysis follows three primary steps—(1) parameter knowledge capture using the prior distribution, (2) determination of the likelihood function, and (3) combination of the prior distribution and likelihood function to obtain the posterior distribution [205].

The prior distribution is determined prior to data collection and can take different forms (e.g., Poisson, normal, uniform, etc.) [205]. Additionally, the prior distribution may have different informative levels ranging from total uncertainty to high relative certainty [205]. Prior distributions are organized into one of three subjective categories based on the level of uncertainty—informative, weakly informative, or diffuse [205]. Hyperparameters are the individual parameters controlling the degree of uncertainty in the prior distribution [205]. Prior distributions are constructed using prior elicitation [205]. Methods for this include consulting an expert to provide the prior distribution hyperparameter values [105, 111, 151, 210], using MATCH, a tool for generic expert elicitation [145], referring to previous publications or meta-analyses [108, 174]. These methods can also be combined or varied as needed [206]. Some reported prior elicitation methods involve the use of data-based prior distributions [205] where hyperparameters are derived using maximum likelihood [35, 42, 43, 207] or sample statistic methods [58, 173, 215]. However, this type of procedure leads to double-dipping [58] since the data set used to determine the prior distribution is also used to calculate the posterior and is not recommended [205]. Alternatives include hierarchical modeling strategies where the prior distribution is dependent on data-driven hyperparameter values [205].

Informative prior distributions are those with high certainty regarding the estimated model parameters [205]. An informative prior distribution is useful when a parameter or relationship between parameters has a known restriction range [205]. However, informative prior distributions can also produce posteriors that do not correctly reflect the population model parameter [205]. Weakly informative prior distributions are less defined and have a hyperparameter with larger variances compared with informative prior distributions [205].

As a result, weakly informative prior distributions have smaller impacts on the posterior, allowing the posterior to be more strongly influenced by data observations expressed in the likelihood [205]. This type of distribution is useful for situations where there is a desired amount of uncertainty about a parameter despite there being some assumed information [205]. Finally, diffuse prior distributions have little to no certainty about the estimated model parameters. Diffuse prior distributions represent a flat density without parameter knowledge [205]. This type of distribution is useful for situations with no certainty of the parameter values, allowing the data to determine the posterior. Diffuse prior distributions can also be implemented as a placeholder or preliminary data collection technique before analyses with other more informative priors are used [205].

While it is important to choose the correct informative level of the prior distribution, mis-specified models or biased data can also prevent the prior distribution from conforming with the likelihood [205]. The likelihood sensitivity analysis can be assessed to check this. Prior predictive checking is used to check the accuracy of the prior distribution before generating any data [205]. A prior predictive distribution can be implemented to show the distribution of the possible samples when the model is true [205]. The more accurate the prior distribution, the more similar the prior predictive distribution will be to the data-generating distribution [205]. In prior predictive checking, the observed data or observed data statistics are compared with the prior predictive distribution or prior predictive distribution statistics to measure their compatibility [205]. Kernel density estimation is a non-parametric smoothing approach used to calculate the probability density function to compare original data with the predictive distribution data [205]. The compatibility of the prior distribution can also be measured using the prior predictive p-value which describes where the observed data characteristics lie within the tails of the reference prior predictive distribution [205]. In the case where two prior distributions are being considered for a model, the Bayes factor is used to compare them [205].

The likelihood quantifies how large of a role the observed data plays in determining possible values for unknown parameters [205]. Bayesian inference refers to unknown parameters as random variables to form probability statements about the unknown parameters [205]. Observed data is considered fixed while parameter values are variables. As such, the likelihood is a function of given parameters, θ , for fixed data y , meaning that the likelihood function summarizes a statistical model stochastically generating the data, a range of possible given parameters (θ), and the fixed observed data (y) [205].

Once the prior distribution and likelihood are defined, data can be collected, and, finally, the posterior distribution obtained. The model can be fit to obtain the posterior distribution using a variety of methods. To do this, the model parameters are assigned probabilities based on their associated uncertainties [205]. Bayesian statistics aims to estimate the posterior distribution of all the model parameters [205]. As a result, the posterior distribution equation is generally known only to a known amount of the proportionality, a set constant that cannot be calculated [205]. Markov chain Monte Carlo methods can be used to indirectly infer the posterior distribution [205].

After the posterior distribution has been determined, the model can be used to produce simulated data. This is only valid if the model can accurately predict data trends. Posterior predictive checking is used to determine the accuracy of simulated data by comparing the estimated kernel density of the observed and simulated data sets [205]. Any form of parameter-dependent statistics or discrepancies can be used to perform posterior predictive checking, and the sensitivity of the test can be varied to suit the user's goal.

Bayesian statistics can be applied to countless applications including social and behavioral sciences, ecology, genetics, reproducibility and data deposition, and limitations and optimizations [205]. Of these applications, Bayesian inference has been especially helpful with respect to active learning, a type of ML where model training begins with a small portion of the labeled data [144]. The primary objective of active learning tasks is to reach a desired accuracy level while reducing labeling costs by making the process of asking for labeled data more efficient [144]. In other words, active learning seeks to improve accuracy while reducing the required amount of labeled data [144]. Most DL solutions, however, are data hungry. This is in contradiction to the small data trends in active learning. Bayesian inference offers a potential answer to this problem.

Non-parametric models like Bayesian statistics-based methods can be applied to model all data types at the cost of higher complexity [144]. As such, DL an effective ML tool, especially for active learning tasks like question answering and NLP problems [144]. Many novel question answering applications have been recently reported that improve performance and effective learning while lowering computing cost and annotation budget [82, 156, 189].

DL methods generally involve the training of convolutional neural networks (CNNs) [144]. Bayesian CNNs are able to learn using only small quantities of data while allow-

ing for representation of model uncertainty with the use of an acquisition function [144]. These qualities make Bayesian CNNs an excellent tool for active learning tasks on big data sets containing high dimensional samples [144]. Bayesian inference-based methods can be used to introduce a probabilistic framework to DL and ML tasks. Bayesian active learning methods offer uncertainty representation along with improved generalization on small quantities of data [144].

Gidiotis and Tsoumakas reported the development of a Bayesian Active Summarization (BAS) [82]. The BAS approach applies active learning to abstract text summarization with the aim to mitigate data dependence of the summarization models [82]. BAS combines DL with Monte Carlo dropout to measure the summarization uncertainty. This is then used to select samples to be annotated [82]. To maximize gains on a limited data annotation budget, annotating and training are iteratively alternated [82]. BAS has improved data efficiency over random selection with higher overall performance for small annotation budgets [82]. Using fewer than 150 training samples, the BAS approach was able to achieve 95% of the performance of the PEGASUS summarization model trained using the full XSum dataset [82]. This suggests implementation of Bayesian inference in NLP can produce more robust learning with reduced computational cost.

Simpson, Gao, and Guervych also implemented a Bayesian approach to text rankings called Bayesian optimisation (BO) [189]. BO acquisition functions present a new potential alternative to the standard uncertainty-focused acquisition functions [189]. BO focuses on maximizing a function while also minimizing the number of queries [189]. In this instance, Simpson et. al. applied BO to a ranking function used to map text documents. They found the BO active learning strategy minimized the number of labels required to identify the best candidate over uncertainty-based methods that focus on learning the complete ranking function [189]. BO aims to identify the best candidate by focusing on strong candidates and disregarding lower quality candidates as opposed to ranking them precisely. This is accomplished by combining pairwise feedback with Gaussian process preference learning (GPPL) [189]. Simpson et. al. reported high accuracy with minimal feedback with improved model matching to human-produced model summaries. As such, BO may be a superior method for language-based tasks like question answering, summarization, and translation.

Bayesian inference has also been applied to passage scoring for open-domain question answering (QA) by Paranjpe, Ramakrishnan, and Srinivasan in the form of a Bayesian

belief network (BBN) [156]. A Bayesian Network is comprised of a directed acyclic graph (DAG) which encompasses a set of conditions about variables in X along with a set of local probability distributions for each variable [156]. Paranjpe et. al. created their BBN using a lexical network where semi-supervised learning was used to simultaneously train the model while associating text tokens to synsets in the WordNet [156]. The inference algorithm was separated from the knowledge base design to ensure the system was extensible and could be trained using a corpus [156]. Once again, Bayesian inference produced strong results using minimal input (i.e., a small corpus). However, one problem with this model was computation and memory costs. In this study, Bayesian inferencing took 0.03 seconds per passage on average, and the memory requirement was up to 30 MB [156].

Gal and Ghahramani refined the general active learning framework by using the kernels of a CNN to behave as the training engine for the active learning framework [78]. By running the Bayesian inference through an approximate inference within the Bayesian CNN, Gal and Ghahramani reported the solution to be more computationally tractable [78]. They also empirically showed dropout as a Bayesian approximation as a means of introducing uncertainty within DL methods [78]. Note that dropout was used during training by applying dropout before each weight layer along with during testing as a means to sample from the approximate posterior [78]. Compared with other active learning models including those using RBF, the Bayesian CNN showed superior performance.

Jedoui et. al. increased the uncertainty level in Bayesian-based models by adding the assumption that the output space is not mutually exclusive [109]. Having multiple outputs for one input would be an example of this scenario [144]. This study reported that classical uncertainty sampling does not outperform random sampling for question answering tasks. Instead, combining Bayesian uncertainty with semantically structured embedding was used [144]. Jedoui et. al. suggested that dropout can be represented by a variational Bayesian approximation [109]. The dropout then becomes a mixture of two Gaussian distributions with small variances where the mean of one of the distributions is equal to zero [144]. Uncertainty within the weights results in prediction uncertainty that could be measured by the approximate posterior via Monte Carlo integration [109].

Pinsler et. al. reported another successful application of Bayesian inference for DL applications [161]. DL was investigated for large data sets and networks for active learning tasks where systematic labeling requests are required for batch active learning tasks [161]. A model focused on efficiently scaled active learning was proposed where the data pos-

terior was well estimated [144]. Different active learning models were considered using various acquisition functions were tested for the efficient batch selection task with sparse subset approximations [144]. Of the tested models, Bayesian active learning produced the strongest performance.

A Bayesian prior has been applied to the network weights within a CNN to prevent over-fitting in CNNs in the form of a Bernoulli approximate variational inference method [79]. This method allows CNNs to learn from small data sets while preventing over-fitting or increase in computational complexity [79]. This is an important development towards adoption of Bayesian statistics in active learning applications. Gal et. al. compared several models and different acquisition functions for NLP tasks [79]. Of the tested models, deep Bayesian active learning showed the best performance.

Overall, Bayesian inference has been established as a promising new method for NLP and ML applications, especially regarding question answering. Bayesian inference is especially unique in that it focuses the bulk of the computing power on identifying the strongest candidates, improving the model effectiveness in cases where a correct answer indicates superior model performance.

Chapter 5: PROPOSED QA METHODOLOGY

5.1 Related Pipelines

ML based reading compression tasks associated with question answering has shown a massive progress in recent times. The most important reason for this is the availability of standard evaluation frameworks such as QACNN/DailyMail [99], SQuAD [168], and Natural Questions [119]. Additionally, recent progress of DL architecture like attention-based and memory augmented neural networks have been a groundbreaking achievement so far [29]. Some QA models like BERT have shown very promising performance in the chatbot domain. However, since SQuAD is reading comprehension focused, QA always needs a context to respond with an answer. This has restricted its use in the commercial chatbot market.

However, due to its great performance, it is very important to get the leverage of BERT-based SQuAD. To be able to achieve an end-to-end chatbot model, there are various pipelines in the market allowing BERT-based model integration for commercial chatbot development. The simple and effective pipeline for document or context retrieval is Term Frequency-Inverse Document Frequency (TF-IDF) [143]. There exist many complex methods, but the simplest, fastest, and most popular information retrieval techniques are frequency-based methods. Such methods are carried out using the most popular approach of Bag-of-Words (BOW). BOW is a representation of text or document by counting the occurrences of words in the given text. The fact that BOW does not account for word order is considered its main disadvantage. This frequency of the word in a particular context can be considered as the feature to make a comparison between documents as well as to train other downstream ML classifier models. Such approach of representing text or document is language independent and is applicable to any type of task with no prior information. However, due to various word variants, frequency counts suffers. Hence, to make it more relevant, several techniques such as tokenization, stemming, or lemmatization can be used. Tokenization involves the splitting of a text or a sentence into individual words, and each split word behaves as a unique token. Assigning each token a unique id helps to make the feature simple and general, which eventually is fed to ML models. Meanwhile, stemming involves the shortening of a word into its root word. For example, stemming shortens the

word “consult”, “consulting”, “consultant”, “consultants”, and “consultative” to its root word “consult”. Lemmatization is a similar technique which is an advanced form of stemming that basically understands the meaning of the word and its context. Its main aim is to shorten words to the base form of word that make sense as a word alone, unlike stemming. For example, lemmatization changes “better” and “best” to its simple form, “good”. Using these preprocessing techniques, TF-IDF assigns weight, increasing the importance of tokens for a specific document. In a document, word frequencies are matched with the ratio of documents in which the words appear, thereby helping to define the basic terms of the document and resulting in a better frequency representation of a text.

Table QA is another a modern platform and interface that will help even non-technology-savvy users to obtain information from transparent datasets without requiring advanced data processing software and without needing to completely understand the nature of the dataset [203]. The pipeline demonstrates the response from tabular data to natural language questions and explores the relevant device setup and model training aspects. The table QA pipeline consists of the following operations:

Learning Table Lookups: The table QA for answering questions from tables is inspired by the End-To-End Memory Network design [193], which is used to turn natural-language questions into table lookups. The Memory Network is a recurrent neural network (RNN) equipped to integrate continuous representations of an input table and a query to predict the correct response. It is comprised of a sequence of memory layers that go over the input table content many times and perform reasoning in multiple steps. The data samples used for training and testing are fed in batches. Any of the data samples consists of the input table, a question, and the correct answer that corresponds to one of the input table cells.

The input tables, questions and responses are inserted into a vector space using a bag-of-words model, which neglects the word ordering. This method is productive to operate on training results, as the terminology for column headers and cell values are disjointed. In the future work, there can be an added benefit of converting to the positional encoding on the real-world data. The output layer produces the expected answer to the input query and is applied as a softmax function in the size of the vocabulary (i.e. it outputs the likelihood distribution over all possible responses, which may be all of the table cells). The network is trained using stochastic gradient descent with linear start to prevent the local minima. The objective function involves minimizing the cross-entropy loss between the expected

response and the true answer within the training set.

Query Disambiguation: The users may refer to columns with terms that vary from the labels used in the table headings, so a fastText model pretrained on Wikipedia is employed to compute similarities between the out-of-vocabulary (OOV) words from the user query and the words in our vocabulary (i.e. to coordinate or ground the query in the local representation). The resemblance is calculated as a cosine-similarity between the word vectors embedded using the pretrained fastText model. The fastText provides continuous word representation, which represents semantic similarity using both the word co-occurrence statistics and the sub-word-based similarity through the character n-grams. For each of the OOV terms, the query disambiguation module chooses the most similar word from the vocabulary at query time and uses its embedding instead. In table QA pipeline, this technique is especially useful in aligning the paraphrases of the column headings. The similarity threshold is empirically discovered and provides optimum precision/recall trade-off on the results.

Implementation: The implementation shows the power of the pipeline trained to answer questions on semi-structured data. The TableQA prototype is introduced as a Flask web framework and is publicly available. The user interface allows one to enter a custom query for a given sample table. The attention weights are visualized by highlighting the related cells in the input table, which gives a perspective on the data patterns learned by the neural network. There is also an additional table, which provides more information about the underlying prediction process. It includes the triple-wise representation of the input table as processed by the neural network, and the attention weights for each of the memory layers separately.

CONQUEST pipeline focuses on the design processes of the NLP engine, query classification method construction, and description of the machine interaction flow [27]. CONQUEST utilizes a machine learning-based mechanism to classify feedback questions to established models derived from Business Information Graphs (EKGs). The confirmation dialog is used to address inconclusive classifications and request mandatory missed criteria. Additionally, CONQUEST evolves along with the question clarification process: these cases identify question trends used as new preparation examples.

The CONQUEST system is composed of both CONQUEST Trainer and CONQUEST Chatbot modules. The Trainer is able to train necessary Template Based Question Answer-

ing (TBIQA) components in the created chatbot. The second is responsible for running the chatbot and using the developed components to have a TBIQA service. The feedback provided by the programmer to the conquest system is comprised of a set of template questions that can be answered using the method in addition to the EKG (ontology + instances) being examined. The domain ontology provides the instance framework, enabling the description of the occurrence or attribute value form depending upon the situational context (properties and connected associations) (properties and linked relationships). A template query whose mechanism provides answering capabilities is called Question Answering Item (QAI).

CONQUEST Trainer: The CONQUEST Trainer module is only implemented offline using the developer. First, two separate indices are constructed during Index Construction—(1) a class index and (2) a property index. Each index contains details regarding the domain ontology schema being reviewed and are of essential to the following implementation steps. The QAIs processing has second training phase divided into three steps:

- Consistency check
- Query pattern parsing and semantic interpretation
- Vectorial representation (QV) construction for a Question pattern (QP)

The third stage involves training the Named Entity Recognition (NER) module. The NER module is responsible for recognizing potential text in a sentence with Context Vector (CV) values. These texts are used to determine the CV for a given input query. NER allows for possible CV values to be defined immediately from the query, removing the need to separately request each CV during the conversation time. More precisely, in CONQUEST, NER is equipped to understand potential literal CV values. CONQUEST uses a basic regular expression mechanism to define numeric type entities. CONQUEST reuses the *dateparser* library to identify data type entities. For string-type literals, CONQUEST classifies a text for its probable owner pair. This is completed using Apache Solr index querying. For instance, if a person's name is queried, all possible instances of the name attribute are retrieved.

The fourth stage is training the question classifier. CONQUEST completes a semantic enhancement step for the input features using CV as a portion of the classifier input (Semantic Features). For the classifier training, the QVs collection generated during QAI

processing is used for the training dataset, using the corresponding QAI for each QV as the classifier output label. The default ML model implemented by CONQUEST is the Gaussian Naive Bayes (GaussianNB) which, when combined with the utilization of semantic features, showed superior model performance in terms of ranking hit rates along with required learning time. The final state is to save the trained artifacts such as Ontology index, the QA items, NLP model, and the classification model.

In addition to these above-mentioned QA pipelines, the development of open-source research resources has been a strong culture in the NLP and ML communities. The BERT original source code [64] and the groundbreaking tensor2tensor library [208], both from Google Research, inspired the structure of Transformers. AllenNLP first reported the idea to provide simple caching for pretrained models [80]. Transformers build on these elements by adding additional user-facing features such as quick model installation, caching, and fine-tuning, as well as a smooth transition to development. Transformers retain some compatibility with available libraries, most notably a tool for inference using Marian Neural Machine Translation (NMT) and Google’s BERT models. Spacy, AllenNLP, flair, and Stanza are examples of general-purpose, open-source libraries that concentrate primarily on ML for various NLP tasks [22, 80, 102, 165]. Transformers are close to these libraries in terms of features. Transformers are connected to common model hubs such as Torch Hub and TensorFlow Hub since it provides a NLP model hub. These model hubs collect framework-specific model parameters for easy application. Transformers, in comparison to these hubs, is domain-specific, allowing the system to automatically help model analysis, use, implementation, benchmarking, and easy replicability.

Transformers are designed to operate in the same way as a typical NLP ML algorithm pipeline; a transformer processes data, applies a model, and then makes predictions. Although the library contains resources to help training and development, we will focus on the primary modeling requirements in this technical report. A transformer located at the library center includes carefully checked applications of Transformer architecture variants commonly used for NLP. There are numerous implemented architectures provided by the Huggingface libraries. They are:

- Language Modeling
- Sequence Classification
- Question Answering

- Token Classification
- Multiple Choice
- Masked LM
- Conditional Generation

Although multiple transformers employ the same multi-headed attention core, they still possess significant differences such as positional representation, masking, padding and sequence-to-sequence design. Various models are also being developed to target a variety of NLP applications such as comprehension, generation, and conditional generation, as well as specialized tools such as quick inference and multilingual applications.

Different architectures use the same API wherever possible, allowing users to quickly switch between models. A collection of **Auto** classes provides a single API that allows for lightning-fast switching between models and frameworks. The configuration defined by the user-specified pre-trained model is used to automatically instantiate these classes. Transformers strive to make the use and delivery of pre-trained models as simple as possible. Inherently, this is a team effort; a single pretraining run allows for fine-tuning on a number of tasks. Any end-user may use the Model Hub to access a model and use it for their own data. Its user interface is supposed to be simple and available to the public. This package can be a model that was trained using the library or a checkpoint from another common training tool. Next, the models are saved and assigned a canonical name that can be used in two lines of code to import, cache, and run the model for fine tuning or inference.

One of Transformers' most significant objectives is to simplify model production implementation. Different users may have different production requirements, and implementation also entails addressing problems that are drastically different from preparation. As a result, the library supports a number of output deployment techniques. One of the library's most significant features is that models are accessible in both PyTorch and TensorFlow, with interoperability between the two frameworks. A model trained using one of these frameworks may be saved using the process of standard serialisation and also can be reloaded using the save files in different frameworks very easily. There are deployment suggestions for each system. Models in PyTorch, for example, are compatible with TorchScript, which is an intermediate representation of a PyTorch model capable of running more efficiently in Python or in a high-performance environment like C++. Models

that have been fine-tuned can now be exported to a production-friendly setting and run via TorchServing. Inside TensorFlow's ecosystem, there are several serving choices suitable for direct use. Additionally, models are able to be exported to intermediate neural network formats for additional compilation using transformers. Finally, as Transformers grow more commonly used for all NLP applications, it is becoming more important to apply them within edge devices like phones and home electronics.

5.2 Implementation in Google Cloud Service

Below is the list of things that are needed before deploying the model into google cloud run:

- GitHub repository (including codes and model)
- Google cloud run account

The GitHub repository is used to push the code to google cloud storage. Therefore, the repository should be updated with all codes and data necessary to run the model. Later, the model can be exposed as a service on the cloud by choosing the deployment option from GitHub. To accomplish this, Google Cloud Build performs continuous integration using GitHub. The mandatory files that the GitHub repository should have are listed below:

- model (saved pytorch_model.bin file, config file, vocab list)
- pipeline (script to predict the answer for given context and question)
- flask servers (serve request)
- docker file (for a running container)
- requirement.txt (for installing required libraries)

The deployment of the QA model in Cloud Run follows the following steps, which are highlighted in Figs. 5.2 and 5.3:

- enable cloud run API (Go to “Marketplace” in Cloud Console and enable it)

- select cloud run from Google Console, as shown in Fig. 5.1.
- create service in a service setting
- select Cloud Run (fully managed) for the Deployment platform
- select Region, followed by service name (after creating service, do not forget to enable “Allow unauthenticated invocations”)
- select SET UP WITH CLOUD BUILD to connect the source GitHub repository (need to select Dockerfile as a build type that will execute a docker container inside Google Cloud Run)
- click on “Show Optional Settings” to change Memory, CPUs, and timeout (recommend using 4GB of memory and 4 CPUs for faster operation)

5.2.1 Implementation in Webpage

The QA webpage is implemented from the cloud build. The webpage is designed using HTML and CSS, which is saved in the “templates” folder in GitHub. The flask app is first configured as in Fig 5.4, which is used to render the web page from the templates. After building the docker, the service URL is generated from the cloud run that can open the web page as in Fig 5.5. The deployed webpage contains two fields: Context and Question. The context is entered in the “Context” field, and question in the “Question” field. Then the submit button can be clicked to get the appropriate answer along with a question.

5.2.2 Implementation in Pepper

Unless we can apply ML to real life, the model is useless. Creating a chatbot is easier than ever. Google’s DialogFlow is an obvious choice because it is effortless, fast, and free. Dialogflow is Google’s Chatbot platform. We will use the Google online console to create a Dialogflow agent that will be able to answer text queries (i.e. users can enter text in the text field, which Pepper will hear). The following steps are needed for chatbot integration with Dialogflow:

- The trained QA model

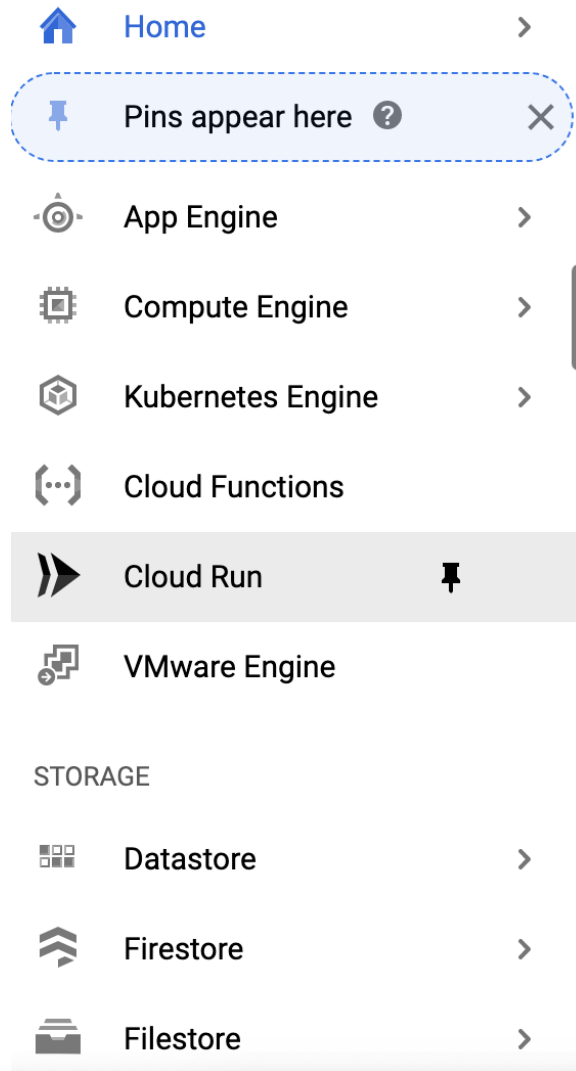


Figure 5.1: Select Cloud run from Google Console.

- The DialogFlow agent, which takes input from a user.
- A Flask app deployed on the web host, which renders the users request and QA response
- Pepper makes a webhook call to the flask API to send the user's request and fetch the response
- Integration of DialogFlow with Pepper Android SDK plugin

A user can access the Pepper chatbot, which will be created in DialogFlow and integrated with the Pepper android plugin later. The conversation will begin, and the chatbot will ask


```

@app.route('/')
def index():
    if request.args:
        context = request.args["context"]
        question = request.args["question"]
        answer = model.predict(context, question)
        return flask.render_template('index.html', question=question, answer=answer['answer'])
    else:
        return flask.render_template('index.html')
    return response

```

Figure 5.4: Flask App for calling the html web page.

The screenshot shows a web application titled "Romanian BERT QA". It has two main input sections: "Enter context:" and "Enter question:". The context input is a text area containing a detailed description of the Samsung Galaxy A31's macro camera. The question input is a text box containing the question "Cum sunt Toate mesajele iMessage pe care le primesc". Below the question input is a green "Submit" button. Below the submit button is a light blue box containing the question "Ce surprinde Macro Cam la Samsung Galaxy A31?". At the bottom, there is another light blue box containing the text "surprinde detaliile din apropiere".

Figure 5.5: Demo of Romanian BERT QA model in webpage.

public host. This API is composed of the application, which will convert the question into a feature vector, fit it to our QA model, and then use the generated answer to respond to the user.

First, DialogFlow is opened, prompting a Google account login. Next, "Create Agent" is clicked to create the chatbot. Next, a design that requests data from the user and es-

establishes a Webhook connection is created. The default greeting intent is then modified to invite the user to choose "yes" or "no". Once the user types "yes", DialogFlow must trigger another intent prompting the user to enter data and save the data point to "entity". In this case, only simple random numbers are being used, so entity creation is not required. DialogFlow has a default unit for managing this data. Therefore, a "yes-Followup Intent" for this sample must be created, because the attempt will be invoked after the user responds positively.

Now, “Add follow up Intent” is clicked to say “yes”. The name of this intent can be changed to another name as needed. Now, the entity must be added, which will contain the data received from the user. For this case, the default unit sys.number is used for a single entry. One parameter for a data point is then created. The model uses a few questions for training to comprehend the desired responses. Next, the chatbot in the right panel is tested to verify whether it is functioning properly. Upon completion, "Webhook for the intent" is enabled to activate compliance. In this way, this special intent will hook the application to the one implemented on the public host Google Cloud Run. Next, the flask application must be compiled and implemented on the Google cloud run, then the URL put in the "Fulfillment" tab on the left. The webhook call is the final implementation stage. Next, the implemented application is connected with the chatbot by entering the URL that implements the application and adding “/webhook”. Remember the flask code of the above application directed to “/webhook”. In the “Fullfillment” tab in the DialogFlow panel (left), “/webhook” is activated, then $\langle your_app's_URL \rangle /webhook$ is added, as shown in Fig 5.6. The webhook response can be checked using a test chat on the right panel.

5.2.3 Integration Dialogflow: Pepper

A Web Chatbot (Dialogflow in this example) can be incorporated into a QiSDK framework for Pepper, and a pre-built Dialogflow agent can be used to create a Pepper that can response to queries with an answer. The requirements for integration are:

- a Pepper QiSDK (NAOqi 2.9)
- Android Studio with the QiSDK plugin
- an internet connection
- a Dialogflow account

Webhook

ENABLED 

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#) specific to the API version enabled in this agent.


| | | |
|------------|--|---|
| URL* | <input type="text" value="Enter URL"/> | |
| BASIC AUTH | <input type="text" value="Enter username"/> | <input type="text" value="Enter password"/> |
| HEADERS | <input type="text" value="Enter key"/> | <input type="text" value="Enter value"/> |
| |  Add header | |
| DOMAINS | <input type="text" value="Disable webhook for all domains"/> | |

Figure 5.6: Integration of webhook into the fulfillment of Dialogflow.

Before beginning the integration process, special credentials are needed to allow the android application to call the Dialogflow agent. To complete this, a Google clouds service account must be logged into, and a key is added, which is downloaded in json format. This is then renamed to “credentials.json”, which will be needed later on. Next, the project is created with two modules in Android Studio.

- “app”, an android application that uses the QiSDK
- “data”, a java library containing all calls to DialogFlow

The app module does not need to know Dialogflow, and the "data" module is not required to know Android. This will allow isolated Dialogflow calls to be tried without any additional installations on Pepper’s tablet.

The following steps are needed to create an android framework and test it with a Dialogflow agent:

1. **Create Android Project:** A new android studio project (with Kotlin, API 23) must be created. After creating a project, the application needs to be robotified as: (File > New > Robot Application). Then, a new module of “Kotlin Library”, called “data” is

created. Inside the module a class called “DialogflowDataSource” is created, and the package name is updated. The “data” module contains build.gradle dependencies as:

```
implementation 'com.google.cloud:google-cloud-dialogflow:0.92.0-alpha'  
implementation 'com.google.http-client:google-http-client:1.29.1'  
implementation 'junit:junit:4.12'
```

2. Add the DialogflowSourceCode code:

The Dialogflow library here is wrapped in a simple method using a string input that returns an input. The code for “DialogflowSourceCode.kt” is illustrated in Fig. 5.7

```
import com.google.api.gax.core.FixedCredentialsProvider  
import com.google.auth.oauth2.ServiceAccountCredentials  
import com.google.cloud.dialogflow.v2.*  
import java.io.InputStream  
  
class DialogflowDataSource constructor(credentialsStream : InputStream) {  
    private val credentials : ServiceAccountCredentials  
        = ServiceAccountCredentials.fromStream(credentialsStream)  
  
    fun detectIntentTexts(  
        text: String,  
        sessionId: String,  
        languageCode: String  
    ): String? {  
        val sessionsSettings = SessionsSettings.newBuilder()  
            .setCredentialsProvider(FixedCredentialsProvider.create(credentials))  
            .build()  
        SessionsClient.create(sessionsSettings).use { sessionsClient ->  
            val session = SessionName.of(credentials.projectId, sessionId)  
            val textInput = TextInput.newBuilder()  
                .setText(text).setLanguageCode(languageCode)  
            val queryInput = QueryInput  
                .newBuilder().setText(textInput).build()  
            val response = sessionsClient.detectIntent(session, queryInput)  
            return response.queryResult.fulfillmentText  
        }  
    }  
}
```

Figure 5.7: DialogflowSourceCode code.

3. Authentication setup:

A “raw” directory must be added where a “credentials.json” obtained from Google Cloud authentication is put.

Integration with QiSDK application:

Next, the android app must be integrated into the QiSDK application, which can then communicate with the DialogFlow. Pepper receives a string input that refers to the human's query or response to return a ChatbotReaction object (i.e. a potential output) in addition to a reply priority. The android application will then analyze all possible reactions, pick, and execute one (depending on the priority). Next, the following will be created:

- a DialogflowChatbot object
- the ChatbotReactions

1. ChatbotReaction: A ChatbotReaction denotes an action implemented by the chatbot; e.g. typically causing Pepper to reply, but reactions are also capable of playing animations, display content, etc. Two primary types of reactions involved–(1)make the robot reply something and (2) what action to take in the event that there is no way to reply. In the “app” module, a new class named “SimpleSayReaction” is created using the code as shown in Fig. 5.8.

```
class SimpleSayReaction internal constructor(context: QiContext, private val answer: String) :
    BaseChatbotReaction(context) {
    private var sayFuture: Future<Void>? = null

    override fun runWith(speechEngine: SpeechEngine) {
        val say = SayBuilder.with(speechEngine).withText(answer).build()
        sayFuture = say.async().run()
        try {
            sayFuture?.get() // Block until action is done
        } catch (e: ExecutionException) {
            Log.e("SimpleSayReaction", "Error during say: %e")
        }
    }

    override fun stop() {
        sayFuture?.requestCancellation()
    }
}
```

Figure 5.8: SimpleSayReaction code snippet.

This block of code is used by Pepper when there is something to reply. Similarly, another code block called “EmptyChatbotReaction” is created, which is invoked when the reply is empty or there is nothing to reply. The code is shown in Fig. 5.9.

2. Chatbot Creation:

An actual chatbot class will be created that will call the DialogFlow library and output

```

class EmptyChatbotReaction internal constructor(context: QiContext)
    : BaseChatbotReaction(context) {

    override fun runWith(speechEngine: SpeechEngine) {}
    override fun stop() {}
}

```

Figure 5.9: EmptyChatbotReaction code snippet.

the correct reaction, although this is dependent on the result. First, “DialogflowChatbot” class is created as seen in Fig. 5.10.

```

class DialogflowChatbot internal constructor(context: QiContext,
                                             credentialsStream : InputStream
)
    : BaseChatbot(context) {
    companion object {
        private val TAG = "DialogflowChatbot"
    }
    private var dialogflowSessionId = "chatbot-" + UUID.randomUUID().toString()
    private val dataSource = DialogflowDataSource(credentialsStream)

    override fun replyTo(phrase: Phrase, locale: Locale): StandardReplyReaction {
        val input = phrase.text.toString()
        val language = locale.language.toString()
        var answer : String? = null
        try {
            answer = dataSource.detectIntentTexts(input, dialogflowSessionId, language)
            Log.i(TAG, "Got answer: '$answer'")
        } catch (e: Exception) {
            Log.e(TAG, "error", e)
        }
        return if (answer != null) {
            StandardReplyReaction(
                SimpleSayReaction(qiContext, answer), ReplyPriority.NORMAL
            )
        } else {
            StandardReplyReaction(
                EmptyChatbotReaction(qiContext), ReplyPriority.FALLBACK
            )
        }
    }
}

```

Figure 5.10: DialogflowChatbot class.

Then in the *MainActivity*, QiSDK lifecycle calls (QiSDK.register etc.) will be run. A “chat” action for chatbot will be created and run in “onRobotFocusGained” as shown in Fig. 5.12.

3. Finally, the chatbot is ready. The Pepper emulator can be started, and the code can be fully run. The graphical interface can be used to interact with Pepper. The demo of the QA model is shown in Fig. 5.12.

```
private lateinit var chat : Chat

override fun onRobotFocusGained(qiContext: QiContext) {
    val credentials = applicationContext.resources.openRawResource(R.raw.credentials)
    val dialogflowChatbot = DialogflowChatbot(qiContext, credentials)
    chat = ChatBuilder.with(qiContext).withChatbot(dialogflowChatbot).build()
    chat.async().run()
}
```

Figure 5.11: onRobotFocusGained code snippet.

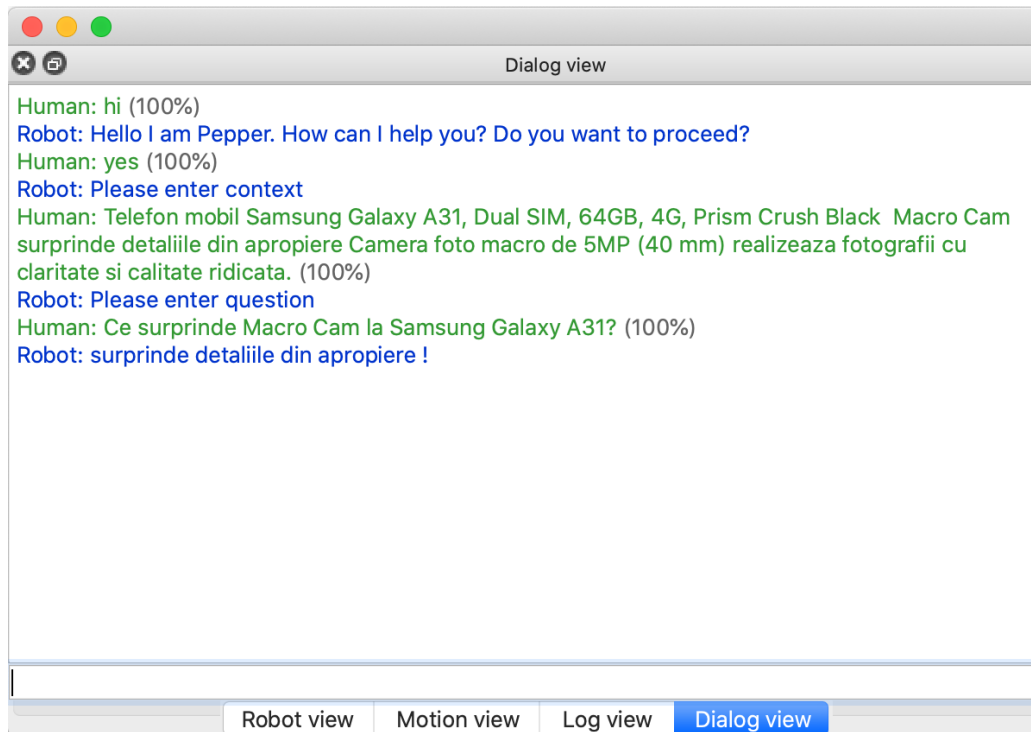


Figure 5.12: QA demo in Pepper.

5.3 Haystack

Haystack is an open-source platform for massive document collections to create end-to-end question answering systems. The application of QA to real world environments has been made possible by recent developments in NLP. Haystack is intended to act as a bridge between science and industry.

Given these above pipelines and architecture, Haystack comes out with major benefits of document retrieval along with integration of pretrained models from Huggingface. Most of the above-mentioned pipelines are knowledge or table-based QA, which fails to integrate the Huggingface transformer library. On the other hand, Haystack provides a complete

framework to extract information using Elasticsearch which helps to retrieve the context for BERT based QA. In addition to this, it also provides framework to integrate Huggingface library for creating an end-to-end chatbot for commercial use.

- **NLP models:** Use all models based on transformers (BERT, RoBERTa, MiniLM, DPR, etc.) and turn smoothly when new ones are released.
- **Flexible databases:** Load data from a number of databases such as SQL, Elasticsearch, and FAISS and query them.
- **Scalability:** Deployments ready for production that scale to millions of documents.
- **End-to-End:** All the tools needed to introduce, analyze, optimize and operate a QA framework.
- **Domain adaptation:** Fine-tune models to own domain and continually enhance them via user feedback.

In order to maximize both speed and accuracy, Haystack is operated by a Retriever-Reader pipeline as shown in Fig. 5.13.

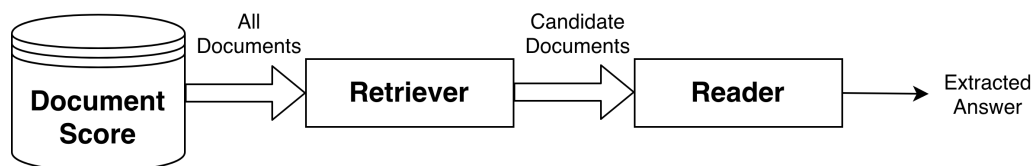


Figure 5.13: Architecture of Haystack [5]

Readers, also referred to in ML as Open-Domain QA systems, are efficient models that interpret documents closely and perform the important primary task of answering questions. The Haystack readers are trained using new language models based on transformers, and GUP acceleration can be implemented to greatly improve upon the speed. Currently, however, it is impossible to directly implement the Reader on large collections of documents. By acting as a lightweight filter that decreases the amount of documents required for processing by the Reader, the Retriever helps the Reader. It does this by following steps:

- Scanning through all the database documents.

- Rapid identification of the relevant and dismissal of the irrelevant.
- Passing only a small number of candidate documents to the Reader.

Uses of Haystack

The uses of Haystack can be extended to various applications of Semantic Search System, Information Extractor, and FAQ style Question Answering.

Semantic Search System

Semantic Search System uses a keyword search in documents to semantically search with Haystack. It is basically achieved by performing question driven queries on stored documents in the database (Elasticsearch, SQL, in memory, FAISS) as shown in Fig. 5.14.

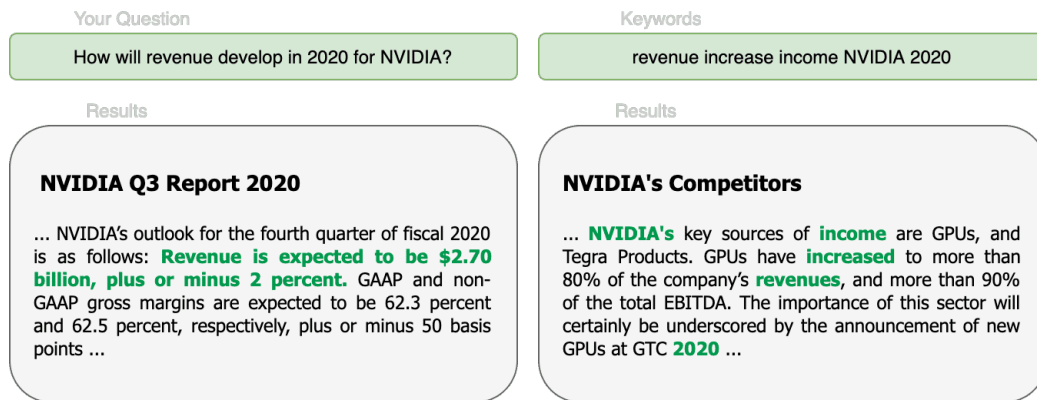


Figure 5.14: Semantic Search System [6]

Information Extractor

It automates the retrieval of relevant information from a set of documents relating to the same subjects, but for various entities. Haystack applies a set of standard question to each of the documents and responses NO_Answer if the particular document does not contain the answer.

FAQ Style Question Answering

It uses existing FAQ documents as well as semantic similarity search to answer new incoming questions. First, it stores the FAQ document in the Haystack, then users represent a new question. Haystack tries to find the closest match the the question in the FAQ database and user will be returned with the most closest question answer pair.

Retrieve Context for QA using Haystack

In order to design a full-fledged chatbot, the QA model of BERT has to be modified. The input for BERT-based QA consists of a question and the context where the answer is present. This dynamic nature of the QA model makes chatbot development difficult. In order to tackle this problem, we use this haystack to automatically search the context for a particular question which is then fed to BERT for the answer as shown in Fig. 5.15. The full step-wise process has been explained below:

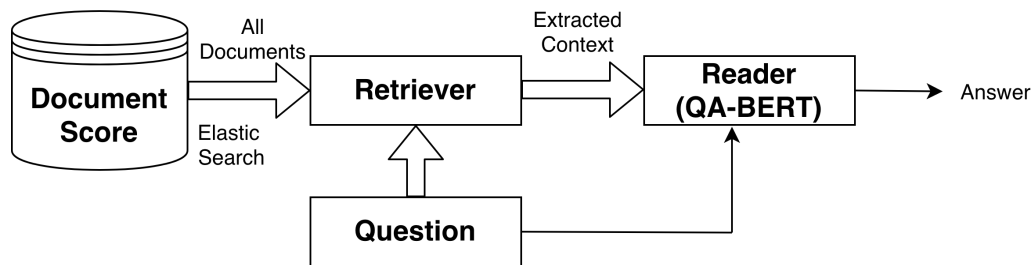


Figure 5.15: Architecture QA model using Haystack.

5.3.1 File Conversion

The haystack uses different converters to remove text from the original files (e.g., docx, pdf, html, txt, etc.). Although the number of styles, layouts, and special cases (especially in the case of PDFs) can be overwhelming, we will discuss the most commonly used formats, including multi-column, and extract meta-information (e.g. page splits). The converters can be quickly expanded to configure them to required files. There are various options for converting files such as Txt, PDF, Docx etc.

5.3.2 Preprocessing

In preprocessing, cleaning and breaking the texts are effective measures that can significantly impact your search speed and accuracy. Splitting larger texts is extremely important in order to achieve rapid query speed. The lengthier the text passed to the reader by the retriever, the slower the queries. The haystack provides a basic *PreProcessor* class allowing:

- Clean whitespace, headers, footers, and empty lines
- Split by words, sentences, or passages
- Option for "overlapping" splits
- Option to never split within a sentence

5.3.3 DocumentStores

This section deals with the preservation of context paragraph in database. The documents can be separated into smaller units (e.g. paragraphs) prior to indexing to improve the granularity and precision of the returned context. We do such operations using ElasticSearch.

5.3.4 Elastic Search

Based on Apache Lucene and developed in Java, Elasticsearch is a distributed, open-source search, and analytics engine. It began as a modular version of the open-source Lucene search platform and then introduced the ability to scale Lucene indices horizontally. Elasticsearch helps in near real-time store, search, and interpret massive amounts of data and return answers in milliseconds very easily. It is capable of obtaining fast search results since it scans an index instead of specifically scanning the text. Instead of tables and schemas, it uses a framework based on documents and comes back with comprehensive REST APIs to store and scan the data. You can think of Elasticsearch at its heart as a server that can handle JSON requests and deliver JSON data back to you. The working of an elastic search can be categorized using the following elements:

Documents

Documents are the simple unit of knowledge that can be indexed to Elasticsearch expressed in JSON. In Elasticsearch, a document can be more than just text; it can be any structured data encoded in JSON. These data may be stuff like numbers, sequences, and dates. Each document has a unique ID and a certain type of data that defines the type of entity that the document is.

Indices

An index is a collection of papers that have similar features. In Elasticsearch, an index is the highest-level object that you can check against. In a relational database schema, you might think of the index as being identical to a database. Usually, all records in an index are linked logically. For starters, in the sense of an e-commerce platform, you will have an index for customers, one for goods, one for orders, and so on. An index is known by a term used to refer to the index when indexing, browsing, reviewing, and deleting operations are conducted against the records it contains.

Inverted Index

An index in Elasticsearch is called an inverted index and is the process used to run all search engines. It is a data structure that stores a mapping of a document or a series of documents from information, such as words or numbers, to its positions. Basically, it is a HashMap-like structure of knowledge that guides you to a document from a phrase.

5.3.5 Retrievers

The Retriever is a very powerful "filter" that can easily run the entire document store along with transferring a selection of candidate documents to the Reader. It is a platform to sift out obviously bad scenarios, stop the reader from performing more work than it wants, and speed up the query process. In our work, we have used the elastic search retriever for fetching a context.

5.3.6 Readers

Readers are mostly Transformer-based models, RomBERT in our case, that read the text in detail to find the answer. We have already trained our dataset using RomBERT, which takes questions from the user and extracts the context for that particular question using Elastic search then find the corresponding answers with their probability scores.

5.3.7 Pipelines

To construct modern search pipelines, dynamic building blocks are required along with a way to combine them together in a flexible manner. The **Pipeline class** is designed used for such a purpose that initiates various search scenarios. A simple example of Open Domain QA Pipeline is given by:

5.3.8 Results

In order to have a QA model, we use a latest Romanian QA dataset known as RoITD [148]. The dataset consists of total 5043 articles with total 9575 questions. The overall statistics of dataset seen in Table 5.1.

| | Train | Test | All |
|-------------------------|-------|-------|-------|
| Number of articles | 4170 | 813 | 5043 |
| Number of questions | 7175 | 2400 | 9575 |
| Average passage length | 52.72 | 61.97 | 55.04 |
| Average question length | 8.08 | 8.12 | 8.11 |
| Average answer length | 13.44 | 7.85 | 9.25 |
| Vocabulary size | 38265 | 18396 | 48821 |

Table 5.1: Dataset Statistics for splitted train and test sets.

| Models | EM | F1-Score |
|---|-------|----------|
| bert-base-multilingual-uncased (mBERT) | 35.48 | 50.94 |
| bert-base-romanian-uncased-v1 (romBERT) | 35.06 | 53.62 |
| distilbert-base-multilingual-cased (distilBERT) | 34.15 | 45.72 |
| ALR-BERT-cased (Romanian ALBERT) | 39.16 | 42.44 |

Table 5.2: RoITD dataset baseline performance.

We then evaluate this MRC dataset using various state-of-the-art models. We employed

two evaluation metrics (i.e., EM and F1-score) to assess MRC model performance on our dataset, consistent with evaluations on English SQuAD [167]. The overall comparison of selected baselines models are shown in Fig 5.2.

- **bert-base-multilingual-uncased (mBERT):** Multilingual BERT (M-BERT) has recently demonstrated surprising cross-lingual abilities, despite being trained without any cross-lingual objectives and without aligned data. mBERT consists of a transformer-based language model that is trained on raw Wikipedia sentences in 104 languages, suggesting an entirely different approach. There is no explicit cross-lingual signal in training, e.g. words, sentences, documents that are linked across languages. In addition to its contextual nature, the model also doesn't require any supervision, since there is no alignment between the two languages. Although mBERT is not explicitly trained with cross-lingual objectives, the representations it produces seem to generalize across languages for a variety of downstream tasks. Here in case of RoITD dataset mBERT achieves EM of around 35% and F1-Score of 50.94%. This is assumed to be the baseline for this dataset. This is because multilingual BERT has been recognized as an established multi language model. This is one of the best performing model of RoITD because with mBERT, the WordPiece modelling strategy can share embeddings across languages. Moreover, mBERT efficiently learns a good multilingual representation with excellent cross-lingual zero-shot transfer performance in a variety of tasks.
- **distilbert-base-multilingual-cased (distilBERT):** distilBERT shares the same general architecture as BERT. As a result of the removal of token-type embeddings and the pooler, the layer count has been reduced by a factor of 2. In the Transformer architecture, most of the operations (linear layer and layer normalization) are highly optimized in linear algebra frameworks, and investigations have shown that variations on the last dimension of the tensor (hidden size dimension) influence computation efficiency less than variations on other factors like the layer count. In the pre-training phase, distillation of knowledge is used, demonstrating that a BERT model can be reduced in size by 40% while maintaining 97% of its language understanding capabilities and being 60% faster. Hence distilBERT performs very decent compared to mBERT. It still manages to achieve almost similar EM with 34.15% but lower F1-Score with 45.72%.
- **bert-base-romanian-uncased-v1 (romBERT):** romBERT is trained using three

publicly available corpora: OPUS, OSCAR and Wikipedia. A vocabulary is built from the corpus available for pretraining. The vocabulary piece count for cased and uncased vocabularies is 50000 using byte-pair encoding (BPE). As a general rule, the better the tokenization of sentences (roughly, the fewer pieces are broken up into each word), the better the model will perform. Romanian BERT can encode a word in 1.4 tokens while mBERT is able to encode a word in up to 2 tokens. A model is trained for 1M steps, with the first 900K trained with 128-byte sequences, and the rest with outperforms all the model including distilBERT and mBERT. It achieves EM of 35.06% and F1-Score of 53.62%. This is the highest F1-Score among the selected baselines. This is due to the fact that romBERT has distinct set of vocabularies that has been language understanding compared to mBERT. This shows that language models must have a strong vocabulary in order to perform well.

- **ALR-BERT-cased (Romanian ALBERT):** ALR-BERT is a multi-layer bidirectional Transformer encoder that shares ALBERT’s factorized embedding parameterization and its cross-layer sharing capabilities. In addition to inheriting ALBERT-base, ALR-BERT-base has 12 parameter-sharing layers, a 128-dimension embedding size, 768 hidden units, 12 heads, and GELU non-linearities. With fewer parameters than BERT, ALBERT achieved new state-of-the-art results on GLUE and SQuAD benchmarks. For achieving cutting-edge results on downstream tasks, it is imperative to have a large network. While BERT is a good candidate for training a large language model on huge corpora, experimenting with large BERT models is difficult owing to memory and computational restrictions. Hence, ALR-BERT gives an alternative to such huge model with very fewer parameters trained on specific Romanian language. The performance ALR-BERT shows that with the reduced parameters, the accuracy does not drop significantly as it achieves F1-Score of only 42.44%. However, it outperforms all the selected baselines for EM as it achieves 39.16%. It shows that despite of having fewer parameters the answers it generates are quite similar to the actual answers. Hence, we can conclude that the quality of answers does not necessarily depends on the size of the model but how efficiently the model is trained.

We now evaluate the execution time of each answer for selected baselines as well as their average EM for selected number of answers. Execution time is calculated in seconds. The comparison of EM and execution time for given baseline models are shown in Fig 5.17.

As we can see that, average EM of several experiments demonstrates that ALR-BERT, bert-base-romanian-uncased, bert-base-multilingual-cased performs quite similar. However, the execution time shows different scenario. ALR-BERT takes maximum execution time compared to bert-base-multilingual. Finally the bert-base-romanian model shows best and least execution time among the selected baselines.

In addition to this, we also study various query used to extract answer from trained model's pipeline. We have used various searching algorithms that are briefly explained below:

- **FAISS:** FAISS is Facebook AI Similarity Search, a library that enables us to swiftly search for multimedia materials that are similar to one another – a problem that typical query search engines fail to meet. Faiss is a library for fast dense vector similarity search and grouping. It includes methods for searching in sets of vectors of any size, up to those that may not fit in RAM. It also includes accompanying code for evaluating and fine-tuning parameters. Faiss has numerous ways for searching for similarities. It is assumed that the instances are represented as vectors with integer identifiers, and that the vectors may be compared using L2 (Euclidean) distances or dot products. Vectors that are comparable to a query vector have the lowest L2 distance or the highest dot product with the query vector. Because it is a dot product on normalized vectors, it also enables cosine similarity.
- **sql:** Structured Query Language (SQL) is a computer language that is used to manage relational databases and execute different operations on the data contained inside them. SQL is used in relational database management systems to edit database table and index structures, add, update, and delete rows of data, and retrieve subsets of information. This data can be utilized for transaction processing, analytics, and other applications that need communication with a relational database. SQL queries and other operations are expressed in the form of statements and aggregated into programs that allow users to add, change, or retrieve data from database tables.
- **elastic search:** Elasticsearch is a distributed, open-source search and analytics engine written in Java that is based on Apache Lucene. It began as a scalable version of the open-source search framework Lucene, then added the ability to grow Lucene indices horizontally. Elastic search allows you to store, search, and analyze massive amounts of data in near real-time and provide results in milliseconds. It is possible

to produce quick search results because, rather than searching the text directly, it searches an index. It has a document-based structure rather than tables and schemas, and it has rich REST APIs for storing and finding data. Elasticsearch may be thought of as a server that can handle JSON queries and return JSON data.

- **In memory:** In-memory analytics is a method of querying data that is kept in a computer's random access memory (RAM), as opposed to data saved on physical drives. As a result, query response times are drastically reduced, allowing business intelligence (BI) and analytic tools to support speedier business decisions. In-memory analytics is becoming more affordable for more enterprises as the cost of RAM falls. Although BI and analytic programs have long allowed data caching in RAM, previous 32-bit operating systems only supplied 4 GB of accessible memory. With up to 1 terabyte (TB) accessible memory (and maybe more in the future), newer 64-bit operating systems have made it feasible to cache massive amounts of data – potentially an entire data warehouse or data mart – in a computer's RAM. In addition to delivering extremely quick query response times, in-memory analytics can decrease or remove the requirement for data indexing and storing pre-aggregated data in OLAP cubes or aggregate tables. This minimizes IT expenses while also allowing for speedier installation of BI and analytic apps.

Given the merits and demerits of search algorithms, we can see that execution time of all selected algorithms are not significantly distinct. Most of them perform similar based on the execution time in seconds. However, Faissembedding performs the worst as it has the highest execution time per answer with 0.786s. inmemorytf stands next to In-memory-tf with 0.785s. The best performing models is inmemoryembedding with execution time of 0.726s. We can clearly see that rest of the model such as elastic search and sql performs very similar to each other with around 0.77s. In all brevity, it can be seen that there are not any significant difference in execution time based on the search algorithms. On the other hand, model type significantly impact the execution time. In addition to this, EM has compelling difference in EM as compared to model type. SQL outperforms all the other search algorithms with EM of 40.35% with second highest EM 39.62% of elasticsearchtf. However, elasticsearchdense performs poorly which is quite similar to faissdense, faiss-embedding, and inmemory search algorithms.

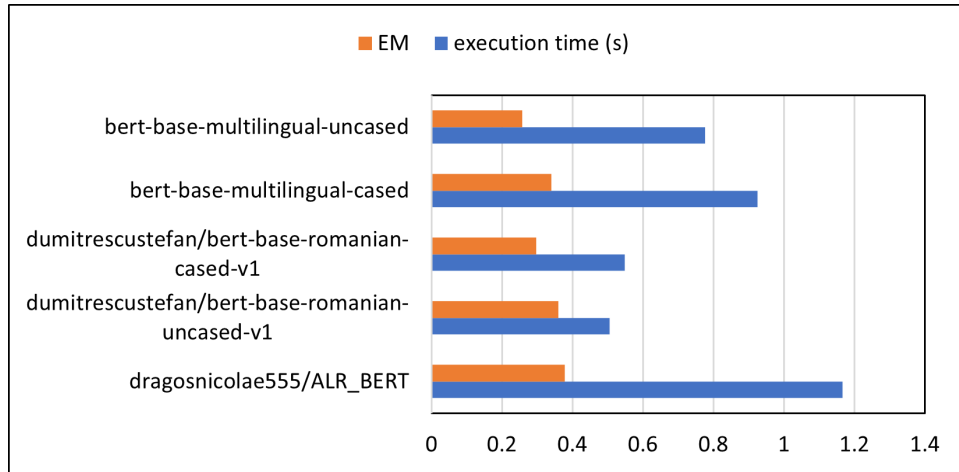


Figure 5.16: Comparison of EM and execution time in seconds for selected models.

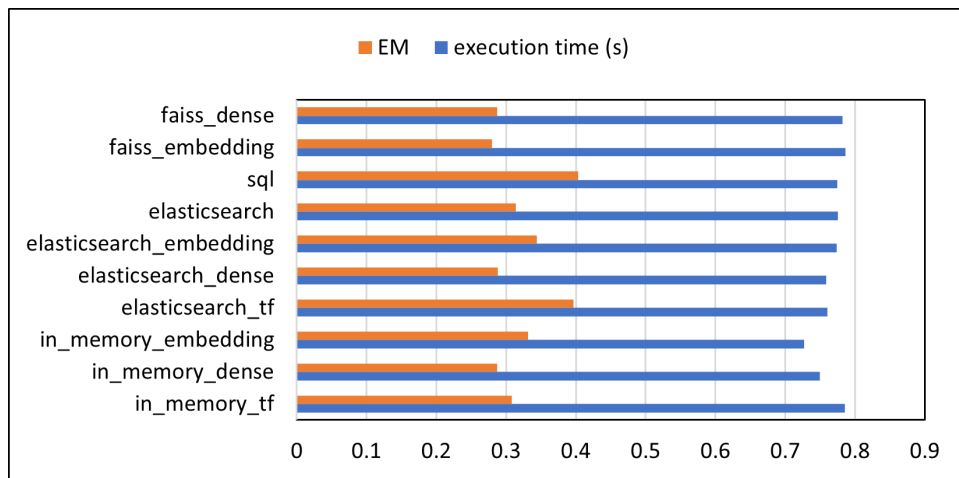


Figure 5.17: Comparison of EM and execution time in seconds for selected models.

5.4 QA using Knowledge Graphs

Question answering (QA) is a particularly challenging natural language processing (NLP) task that has served as a huge source of scientific interest. To answer a given question, models must search through a vast reservoir of pertinent information, then form an appropriate answer that fits within the context of the original question. Additionally, it is expected that this complex language-based task be completed with human-level accuracy. Two general approaches have been developed for QA tasks—implicit encoding and explicit representation [219]. During implicit encoding, knowledge is stored within large LMs by pre-training using unstructured text [41, 160]. In the case of explicit representation, knowledge is contained within structured KGs using nodes to represent entities and

edges to represent the relations between nodes. Examples of KGs include Freebase [39] and ConceptNet [190]. Although LMs can demonstrate broad knowledge coverage, they tend to struggle with structured reasoning such as handling [114]. KGs, meanwhile, show strong performance in structured reasoning [170, 171] to introduce explainable prediction capability [130] but are noisy and deficient in coverage [40, 91].

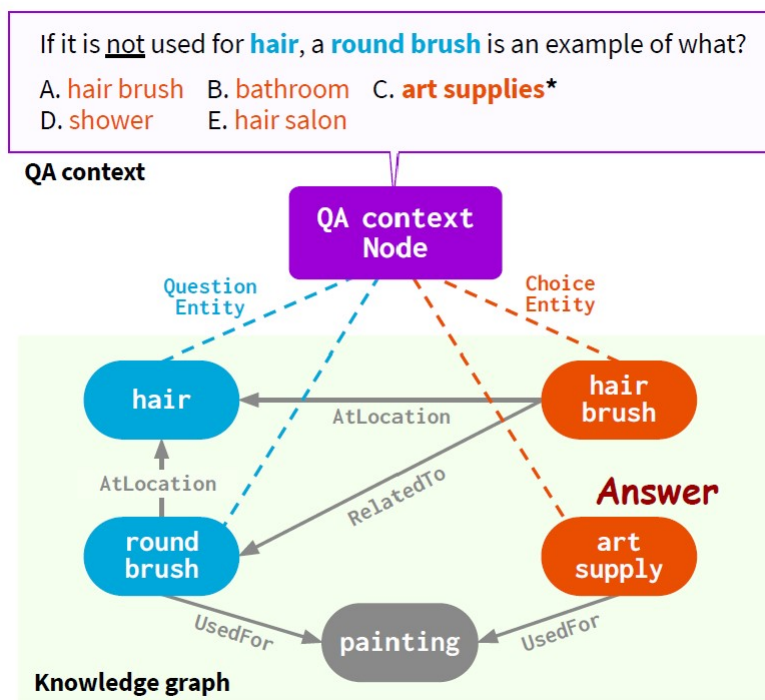


Figure 5.18: Joint reasoning between the language model and knowledge graph (green box) for a given question-answer (purple box) [219]

Although QA procedures have undergone a substantial amount of advancement over the past decade, the most accurate current methods include a combination of LM and KG. To combine these approaches, the model must be able to identify information from a large KG, then perform joint reasoning between the QA context and KG structure (Fig. 5.18). Several of these methods have been reported in the literature. In several of these combined LM and KG models, a subgraph is retrieved from the KG using the KG entities provided by the posed question context, or topic entities, and their few-hop neighbors [30, 130, 194]. A primary disadvantage of this method is that it introduces many semantically irrelevant entity nodes with respect to the QA context, particularly with increasing numbers of topic entities or hops. Other reported combined LM and KG models targeting reasoning applications approach the question context and KG as separate modalities [74, 130, 135, 214]. QA-GNN is a form of combined LM and KG model that uses an end-to-end approach where

the LM identifies the relevant information contained within large KGs, then provides a concise, natural answer [219]. QA-GNN is an end-to-end hybrid LM-KG model designed specifically for QA tasks where the LM encodes the QA context followed by KG subgraph retrieval (Fig. 5.19 [219]). QA-GNN handles QA tasks using a combination of relevance scoring and joint reasoning. Relevance scoring is completed by estimating the importance of each KG node with respect to the provided QA query [219]. Joint reasoning involves the connection of the question context with the KG to produce a joint graph that combines these two modalities to form one working graph [219]. Representations can then be updated using graph neural networks.

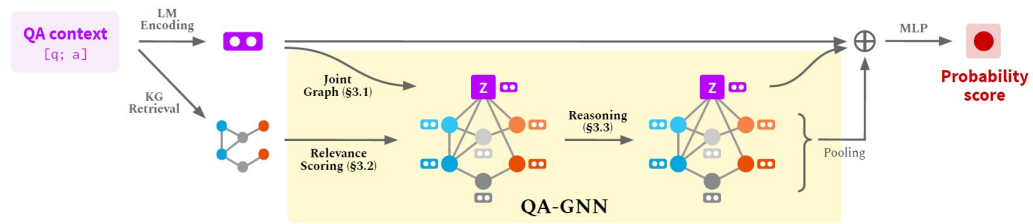


Figure 5.19: Overview of the QA-GNN model where a provided QA (z) is connected with a sourced KG to create a joint graph, also referred to as a working graph. The relevance for each node in the KG is conditioned on z , and reasoning is performed on the working graph. [219]

Domain gaps are a common issue in neural networks and LMs where the model is able to provide human-level performance on the dataset used for training but cannot maintain this performance level when generalizing on other datasets [70]. Transfer learning targets this issue by bridging the gap between LMs and other source domains to improve model performance even in complex target domains [70]. Domain adaptation is a type of transfer learning that minimizes gaps separating target and source domains in order to reduce the amount of data required for model training [70]. Essentially, domain adaptation improves model performance on large datasets other than the training dataset, eliminating the need for additional training aside from only the last layers. In domain adaptation, the bridged datasets generally have similar classes although they will come from different distributions that create large style differences. KGs are used to focus the LM and incorporate large volumes of information other than the pretraining data.

KGs can be used for a vast variety of data forms including digital humanities [71, 106, 139, 184]. KG embeddings (KGEs) are used to map the nodes and entities of a specific KG component into a low-dimensional continuous vector space in order to simplify the

KG data [70]. The method used to complete this process uses a score function to measure the distance between two nodes within the mapped embedding space. Thus, the score function maintains the position of nodes within the KG despite this data simplification. KG embedding can be completed using two primary components—an encoder and a classifier (Fig. 5.20) [70]. The classifier uses a dimension mapper, or latent space mapper, to convert the dimensions of the visual embeddings to match the embeddings in the KG (Fig. 5.20). The KG embeddings form the final encoder layer. The source domain is derived from the original KG used to connect dataset information with the concepts, C^S , contained within the KG to form data clusters around the concepts. This allows for information linkage with class information along with several axes. The source domain is described by $X^S = (x_i^S, y_i^S)_{i=1}^N$ where x_i^S are the variable sized input images, y_i^S are the associated classes, and N is the source dataset size [70]. The target domain data is described by $X^t = (x_i^t, y_i^t)_{i=1}^M$ where y_i^t is the respective class, and M is the target dataset size [70]. The model function, f , is comprised of the encoder function, e , and classifier function, c , where $f = e \circ c$. In this relationship, $e : X \rightarrow Z$ maps input information to a 1D vector latent space [70]. This is the embedded information that is later classified into a corresponding class where the embedding space is mapped into the label space which is described by $c : Z \rightarrow Y$ [70].

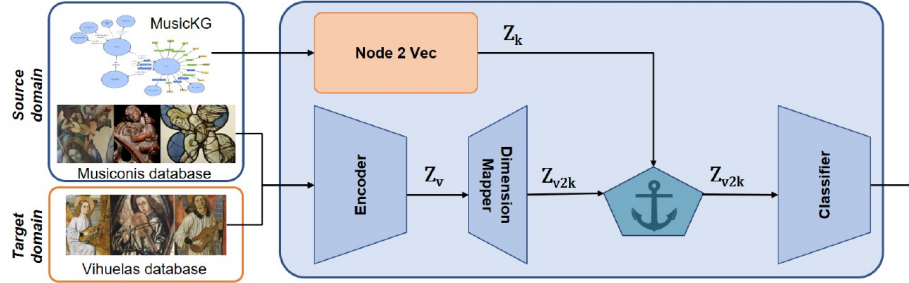


Figure 5.20: Algorithm details of knowledge graph embedding-based domain adaptation. [70]

The encoder transfers the input shape from a 3-channel 2D array to a 1D vector accessible to LMs, as described by the following equation:

$$z_v = E(x), x \in X^s \cup X^t, z_v \in R^v \quad (5.1)$$

where $z_v \in Z_v$ annotates the 1D vector extracted using the pre-trained encoder that extracts information from the KG [70]. A set of v float numbers within the embedding

space can be described as $v \in [256, 4096]$, although this is model dependent [70]. The output produced by the encoder is passed to the spacemapper used for classifier training. The embeddings are class-informative (i.e., extracts information about the object label) and domain-independent (i.e., same-class information is grouped together independent of their domain origin). The source domain output from the encoder is represented as $z^S = E(x^S)$. The latent space of the target samples is represented as $z^t = E(x^t)$.

The dimension mapper deals with the variable latent space dimensions produced by the pre-trained model by outputting encoder embeddings in R^v to match the dimension of the KG in R^k . This is described by the following equation [70]:

$$z_{v2k} = DM(z_v) = DM(E(x)), x \in X^s \cup X^t \quad (5.2)$$

The dimension mapper function maps values according to $R^v \rightarrow R^k$.

The classifier acts as a simplified version of the fully-connected LM and is most commonly used for image classification. For a cross-entropy loss function, the classifier input is the mapping of the visual latent space that is transformed from R^v to R^k with $k < v$ [70]. The output is comprised of a 1D probability vector. This serves as the model's prediction along with the probability of it belonging to each class, \hat{y} . The following equation describes the classifier function:

$$\hat{y} = C(z_{v2k}) = C(DM(E(x))), x \in X, X = X^s \cup X^t \quad (5.3)$$

where \hat{y} is the per-class probability vector. This is the classifier output shared between both domains as described by $\hat{y} \in \hat{Y}$, $\hat{Y} = \hat{Y}^S \cup \hat{Y}^t$ [70]. The encoder, dimension mapper, and classifier are trained on the source domain until convergence is achieved prior to beginning domain adaptation. The output from the classifier regarding the target domain is later used as pseudolabels that can be used to improve the model training provided the classifier is confident in the process.

Several loss functions are used to train the model and its components including classification loss and anchoring loss. Classification loss is the first loss function used to train the model and impacts encoder, classifier, and dimension mapper training. Differences between

the output and label probability distributions are reduced by the cross-entropy loss function $H(.,.)$ using the following equation [70]:

$$L_c(W^E, W^C, W^{DM}) = (1 * \sum_{x^S \in X^S} H(\hat{y}^s, y^s) + \lambda_t \sum_{x^t \in X^t} H(\hat{y}^t, y^t)) \quad (5.4)$$

where λ_t is a hyperparameter used to control the loss contribution shared by the two domains. The anchoring loss marks the KG embeddings as anchors, then moves mapped embeddings closer to these anchors to improve their quality. This embeds additional information in the encoder training without requiring the use of the data as classifier input [70]. The anchoring loss combines increased fusion model accuracy with the superior speed and generalizability provided by a single source model [70]. Linear discriminant analysis (LDA) and Fisher's linear discriminant form the anchoring loss which is described by the following equations:

$$L_{anc}(W^E, W^{DM}) = \left(\frac{\sum_{i \in Y} \sum_{c \in C} (d(\mu_{v2k}^c, z_{v2k-i}^c))}{\sum_{ci \in C} \sum_{cj \in C} d(\mu_{v2k}^{ci}, \mu_{v2k}^{cj})} \right) \times \lambda_{BF} \quad (5.5)$$

$$\lambda_{BF} = \frac{\min_i |Y_i^t|}{\max_i |Y_i^t|} \quad (5.6)$$

where μ_{v2k}^c is the center of the mapped visual concept embedding, a_k^c is the corresponding anchor, $(d(\mu_{v2k}^c, a_k^c))$ is the reduced distance between the mapped embedding and corresponding anchor, z_{v2k-i}^c is the distance between the center of the mapped visual concept embedding and the corresponding mapped embedding, and $(d(\mu_{v2k}^c, a_k^c))$ is the reduced distance between the mapped embedding and the corresponding mapped embedding. The anchoring loss targets the minimization of both of these distances. One disadvantage of the anchor loss is that it generally produces higher values in comparison with the classification loss (by approximately 10-15 times), which can influence the classification direction. Additionally, this loss has a tendency to be imbalanced and depend on random samples that requires the use of a balancing factor λ_{BF} . The losses can be optimized using:

$$L = \min_{W^E, W^C, W^{DM}} 1 * L_C + \beta_A L_{anc} \quad (5.7)$$

where β_A is a balancing parameter (usually between 0.01-0.03).

```

Algorithm 1: Knowledge Embedding based Domain Adaptation.

Input:  $X^s$ —Source domain images.
 $Y^s$ —Source domain image labels.
 $KG^s$ —Source domain knowledge graph.
 $X^t$ —Target domain images.
 $Y^t$ —Target domain image labels.
 $\beta_A$ —Balancing factor—hyperparameter

Output:  $\theta^E$ —Weights of the Encoder
 $\theta^{DM}$ —Weights of the domain Mapper
 $\theta^C$ —Weights of the classifier

// Creating the anchor embeddings  $a_k^c$  using node2vec
Sample walks using a random walk from the  $KG^s$ . ;
Embed the nodes of  $KG^s$  using the skip gram model. ;
Generate the  $a_k^c$  as the mean of the art work embeddings related to the concept;
// Pre-training The Encoder, Mapper, and classifier on the source
domain.
for  $i \leftarrow 1$  to epochs do
    for  $j \leftarrow 1$  to nb_batches do
        Sample a batch of source images  $(x_{1s}^j, y_{1s}^j), (x_{2s}^j, y_{2s}^j), \dots, (x_{Ns}^j, y_{Ns}^j)$ ;
         $\theta^E = \theta^E - \alpha \frac{\partial \mathcal{L}_C}{\partial \theta^E}$  Equation (4);
    end
end

// Anchoring the source visual concepts and adapting to the target
for  $i \leftarrow 1$  to  $I$  do
    Sample a batch of images for both domains  $(x^s, y^s, c^s), (x^t, y^t)$ ;
    Update  $W^E$  by deriving  $\mathcal{L}_C + \mathcal{L}_{anc}$ ;
    Update  $W^E$  by deriving  $\mathcal{L}_C + \mathcal{L}_{anc}$ ;
    Update  $W^C$  by deriving  $\mathcal{L}_C$ ;
end
return  $\theta^E, \theta^C$ 

```

Figure 5.21: Overview of the algorithm. [70]

Figure 5.21 shows a detailed summary of the algorithm used for KG embedding in domain adaptation [70]. First, a KG describing the source dataset is leveraged. Next, an embedding of each datapoint is created based on nodes and the connections between them. The LM is then trained using classification loss. Finally, the encoder is able to extract the structured and class-informative latent space. This allows the classifier to better generalize to other domains.

5.4.1 Results

The objective of this section is to evaluate the effectiveness of incorporating knowledge graphs into the framework of question answering. Knowledge Graphs (KG) are graphs that encode relations of a certain body of text into the connections of entities (nodes) and

relationships between them (edges). To produce this graph, it's necessary to perform a semantic analysis over the text that extracts the fundamental semantic structure composed of *Subject + Predicate + Object* in which the object and subject constitute nodes and the predicate constitutes the edges. In the same sense that it was desirable to produce geometric representations of words for NLP, it is desirable to produce geometric representations of the KG. In this case the most widely spread method is to compute embeddings of nodes and edges in such a way that the nodes that were close together in the graph, are kept close in the embedding space. In this sense, the following relation holds for to arbitrary nodes:

$$\mathbf{v}_{node_1} = \mathbf{v}_{node_2} + \mathbf{v}_{edge_{12}} \quad (5.8)$$

The algorithm producing the embeddings should try to minimize the distance between nodes 1 and 2 while maintaining the overall graph structure.

The main steps taken in the implementation approach are:

1. First, a series of tuples containing $(context, question, answer)$ are sampled from the data.
2. The encoder-Decoder Model with attention is trained with a Categorical cross-entropy loss function over that data set.
3. The same data-set is also used to compute the knowledge-graph and its embeddings.
4. Another, smaller, data set is then sampled and training is again performed but now the anchor loss function is included to the training process. G.

For computing the knowledge graph incoming from the text corpus the python library [12] was used to make the semantic analysis required for entity and relation extraction. The graph construction was then done using the well known Python library [7] from the lists of source, target and relationship coming from [12]. The next step is to evaluate the Question Answering model, this is usually done by computing a series of metrics which elaborate on distinct goals (final and intermediate) for the model to accomplish. In this project we will focus on two metrics: Exact match and F1 score. These two metrics are tailored to evaluate the model's end result, this is, how well it answers the given questions. The metrics are

calculated as follows:

$$EM = \frac{\text{exactly matching answers}}{\text{total evaluated questions}} \times 100 \quad (5.9)$$

$$\begin{aligned} P &= \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \\ R &= \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} \\ F1 &= 2 \frac{PR}{P + R} \end{aligned} \quad (5.10)$$

We can now present the results of the implementation and contrast them with another very well known framework [2]. Given the available resources we will concern ourselves with presenting the values for the EM and F1 metrics for both models trained over the same data set. The results are shown in table 5.3. It can be seen from them that the model struggles with the dataset as the implementation of the anchor loss approach poses some challenges in order to achieve good performance. The original anchor loss was posed for a classification problem. Superficially, Question answering isn't a classification problem but it encloses one in the form that the last layer of the decoder is interpreted. In this sense, the anchor loss should be reinterpreted to account for the fact that the output of the model is a sequence of logits over the vocabulary space predicting each word of the output sequence. There are two ways one could calculate this loss:

1. Just use the output sequence predicted by the training sampler and use those sequences to calculate the distance between them and the KG embeddings.
2. Using the output logits to calculate a series of predictions and then compute distance between them and the embeddings.

Another challenge that was encountered by the author is the limited access to hardware resources. Training NLP models is a very hardware intensive task and this is reflected in the quality of the results which are. Better hardware is paramount for helping with improving the quality of the model since it will allow us to add richer internal states for the RNN and better word embeddings trained on larger datasets.

Table 5.3: Results for the EM and F1 metrics

| | EM [%] | F1 [%] |
|-------------------|--------|--------|
| Anchor Loss Model | 35 | 25 |
| Haystack | 55 | 54 |

Chapter 6: QA USING BAYESIAN INFERENCE

6.1 Next best Answer using Bayesian Inference

Deep learning multi-class classification models typically produce a probability distribution $\{\pi_i \in \Pi; \sum_{i=1}^K \pi_i = 1\}$ over a set of possible K classes $c_i \in C$. Training a neural network is equivalent to maximizing the likelihood of the parameters θ of the model to describe the data $\mathcal{L}(\theta|X)$. In other words, the training process is an evidence acquisition process where training data is utilized to update the prior knowledge of the model about the problem. During inference, the posterior predictive probability $P(x^*|\theta)$ uses the acquired knowledge about the problem, modelled into the updated prior, to make predictions about new data. Upon the arrival of new observations, updating the prior requires fine-tuning the model by resuming the training process on the newly observed data. However, this can be a costly process when the model is large, which is typically the case with models used in complex problems like NLP where transformer models usually involve millions of trainable parameters. The approach described in this documentation provides an alternative way to calibrate model predictions with new instance-specific observations.

Given a probabilistic output of a machine learning multi-class classification model \hat{p}_i , and an observation about one or more of the events classified by the model y_i^* , the goal is to calibrate the output with the new observations, in a bayesian context, and without retraining or fine-tuning the model. Table 6.1 illustrates the problem, where $\hat{\pi}$ are the classifier output probabilities, y^* are the new observations, and π^* are the updated probabilities, which are the objective.

Table 6.1: An Illustration of the problem.

| | Class A | Class B | Class C | Class C |
|-------------|---------|---------|---------|---------|
| $\hat{\pi}$ | 0.5 | 0.1 | 0.3 | 0.1 |
| y^* | 0 | - | - | - |
| π^* | ? | ? | ? | ? |

We assume events follow a Multinomial distribution, since the output is a single event.

$$x_1, x_2, \dots, x_k \stackrel{iid}{\sim} \text{Multinomial}(n, \theta_1, \theta_2, \dots, \theta_n)$$

where $\sum_{i=1}^k \theta_i = n$, k is the number of possible events, and n is the hyperparameter quantifying the number of trials.

The probabilities of the multinomial distribution are assumed to follow a Dirichlet distribution, since Dirichlet is a conjugate prior to the Multinomial likelihood of the observed events.

$\theta_1, \theta_2, \dots, \theta_n \stackrel{iid}{\sim} \text{Dirichlet}(\alpha_1, \alpha_2, \dots, \alpha_n)$ where α_i is a hyperparameter.

Now that we have the likelihood and the prior, we can easily calculate the posterior:

$$\begin{aligned} p(x|\pi) &= \text{Mult}(x|\pi) \text{Dir}(\pi|\alpha) \\ &\sim \prod_{k=1}^n \theta_k^{x_k} \prod_{k=1}^n \theta_k^{\alpha_k-1} \\ &\sim \prod_{k=1}^n \theta_k^{x_k+\alpha_k-1} \\ &= \text{Dir}(x + \alpha) \end{aligned}$$

Since observations are rather outputs of a softmax function, we normalize the multinomial parameters by n to get probabilities $\pi_i = \theta_i/n$.

Given that new probabilities are provided, how do we update the the probabilities over the rest of the classes?

- We approximate the Dirichlet probabilities to Multinomial counts by scaling them a positive real constant C to approximate them to counts. The constant C is a hyperparameter inversely proportional to the dependency on the prior α of the Dirichlet distribution. Larger values of C correspond to lower importance of the prior. $x_i = C * \pi_i$ where x_i is a positive count.

- We update $\hat{\pi}_i$ with the new observed counts x_i^* :

$$\hat{\pi}_i = \frac{\alpha_i + x_i^*}{\sum_{j=1}^K \alpha_j + x_j^*}$$

We setup our experiment by training ALR_Bert QA model on the RoITD dataset. The tokenizer model used in the experiment is the bert-base-romanian-cased-v1 model. The ALR_BERT model was trained for 5 epochs, which accounted for 5,665 training steps. The RoITD dataset was split into 9,057 samples for training, and 100 for testing.

Our hypothesis is that, suppressing the top prediction and updating the remaining tokens' probabilities would force the model to explore more answers, which in turn is likely to improve the recall of the model.

To evaluate our approach, we apply a softmax function on the model's output logits. The hyperparameter C in the bayesian step is set to the number of tokens in the input. The highest probability is replaced by a 0, and passed through the bayesian update step. Upon obtaining the predicted start and end indices, the correct and predicted answers are processed through a string normalization function. Remove articles, punctuation, and white-spaces, and case-folding the answers is necessary before matching the correct with the predicted answers. Two answers could mean the same thing while not being an exact match. Since it is less likely that the predicted answer matches the correct answer character to character, we abstract the answer into a set of tokens and consider the intersection between the two sets of tokens for performance evaluation. With this configuration, the following classification metrics were used to evaluate each answer pair, and averaged over the entire testing set.

$$\mathcal{P}_{\text{Precision}} = \frac{\text{Number of Common Tokens}}{\text{Number of Tokens in the predicted answer}}$$

$$\mathcal{R}_{\text{Recall}} = \frac{\text{Number of Common Tokens}}{\text{Number of Tokens in the true answer}}$$

$$\mathcal{F}1 = 2 \cdot \frac{\mathcal{P}_{\text{Precision}} \cdot \mathcal{R}_{\text{Recall}}}{\mathcal{P}_{\text{Precision}} + \mathcal{R}_{\text{Recall}}}$$

The fine-tuning performance of the pre-trained ALR BERT model are shown in table ??.

Results in table 6.2 show that recall is noticeably improving after a single bayesian update over the predicted probabilities of start and end scores. We ran an independent t-test to evaluate the significance of the improvement over the recall, and to see if the noticed drop in precision and f1-score are significant. The P-values are not low enough to conclude that there is a significant change in the performance of the model before and after the bayesian update step.

When an expert marks the first predicted answer as wrong, the model is expected to

| Step | Training Loss |
|-------|---------------|
| 500 | 3.76 |
| 1,000 | 3.715 |
| 1,500 | 3.54 |
| 2,000 | 3.47 |
| 2,500 | 3.40 |
| 3,000 | 3.28 |
| 3,500 | 3.30 |
| 4,000 | 3.12 |
| 4,500 | 3.17 |
| 5,000 | 3.01 |
| 5,500 | 2.999 |

Table 6.2: Effect of a single Bayesian update on the accuracy of the 1st answer.

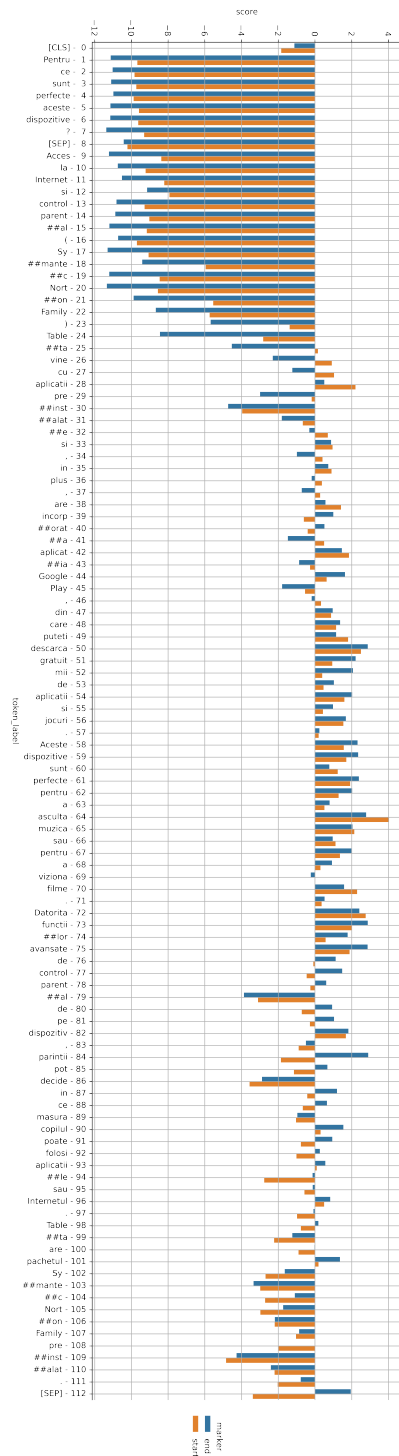
| | Bayesian Updates | | | | |
|-----------|------------------|-------|-------|-------|-------|
| Metric | 0 | 1 | 2 | 3 | 4 |
| Precision | 27.84 | 24.75 | 25.66 | 19.85 | 24.24 |
| Recall | 13.03 | 15.96 | 13.77 | 11.93 | 13.20 |
| F1 | 15.53 | 15.35 | 14.90 | 12.47 | 14.11 |

report the second most probable answer. We evaluate the performance of the bayesian update step on the second most probable answer if the precision of the first answer is less than 90%. Results shown in table 6.3 indicate a promising potential for the bayesian update method to improve the accuracy of the second most probable answer, if the first answer is rejected. Despite the improvement not being significant, it is worth noting that the marginal improvement in accuracy is at negligible computational cost compared to the expensive process of fine-tuning a pre-trained deep model on new observations (i.e. using the predicted first answer as a negative sample).

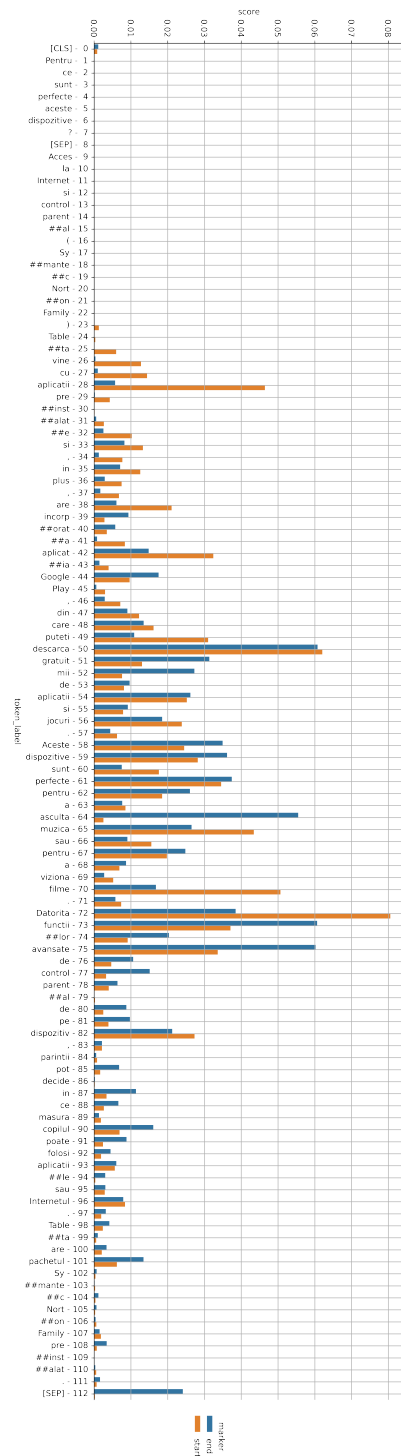
| Metric | No B. Updates | 1 B. Update | P-value |
|-----------|---------------|-------------|---------|
| Precision | 29.02 | 30.14 | 0.58 |
| Recall | 13.78 | 15.37 | 0.32 |
| F1 | 14.34 | 15.98 | 0.29 |

Table 6.3: Effect of a single Bayesian update on the 2nd answer if 1st answer is rejected.

For more granular contrast of the results, we show in figure 6.1 case the difference in the token scores before and after one step of bayesian update.



(a) No Bayesian Update.



(b) One-step Bayesian Update.

Figure 6.1: Token scores with vs without bayesian update.

Chapter 7: CONCLUSIONS

7.1 Conclusions

Recent advances in artificial intelligence have produced significant advances in natural language processing (NLP). The majority of big corporations are particularly interested in NLP applications for question answering (QA) jobs. A entails a user submitting a natural language-formatted query, which the language model then answers clearly and properly (LM). NLP technologies, such as chatbots, should be able to discern intent while dealing with user inquiries. Hence, in this project we design a model that is capable of interpreting questions in real-time using less training data. When a user says anything, the chatbot should be able to conduct intent classification to determine the meaning of the statement. Chatbots, like human conversation, frequently speak in rounds. The goal of the challenge is to accurately answer to your request, since the dialog serves as the context. In this scenario, obtaining the proper answer entails collecting a huge number of small text fragments and categorizing them based on their assessed significance for the inquiry. As a result, the job is to retrieve these text fragments and categorise them in order to answer the question while keeping the context of the dialogue in mind. The primary goal of QA systems is to create technology that not only finds the proper documents but also provides accurate responses to natural language inquiries. More advanced text processing than what is currently used in the information retrieval system is necessary to answer inquiries in natural language.

The application of language modeling to NLP has led to the development of advanced tasks such as machine reading comprehension (MRC). To answer offered queries, MRC requires computational comprehension of natural language content. Achieving this task is especially important for advanced search engines and intelligent agent frameworks, but it comes at a cost of expensive datasets that are difficult to collect and are often limited to one language, like English. SQuAD1.1, SQuAD2.0, and CoQA are examples of datasets used for English Question Answering models. Some alternative datasets have been developed in languages such as Chinese (such as span-extraction MRCs and user-query-log-based DuReaders), Korean, and French. Hence, we use a recently introduced Romanian QA dataset known as Romanian IT Dataset (RoITD). RoITD is a completely Romanian natural language dataset with 9575 QA pairings created by crowd workers. The QA pairings

were created using 5043 Romanian Wikipedia pages about IT and home products. 5103 of the questions are 'possible,' which means the right answer is within the offered paragraph, while 4472 are 'not possible,' which means the provided response is only probable and cannot be regarded accurate. The dataset was assessed by fine-tuning the QA model using the transformer-based pretrained models (1) Multilingual BERT, (2) DistilBERT, (3) Romanian BERT (RoBERT), and (4) XLM-R.

In addition to using the dataset, we extend the RoITD dataset beyond question answering task to formulate a domain classification based on the type of question known as Question classification task. It aids in restricting the search area to a certain sort of query, resulting in a lot better response in QA tasks. The dataset is extracted based on the type of questions such as “Ce”, “Cum”, “De ce”, “Care”, “Unde”, “Când”, and “Caror”. The designed dataset is evaluated using various interpretable models such as SVM, Naive Bayes, Decision Trees, Logistic Regression, and Tsetlin Machine. It gives the advantage of having feasible QA model for chatbot application building a prerequisite question classification model.

In addition to this, we also used outlier detection to make the model robust. Machine learning (ML) models provide enormous versatility due to their ability to learn from a wide range of data sources, including structured and semi-structured data. This learning flexibility, however, necessitates that the model is able to cope with outliers, which in this case are data points that deviate significantly from the training dataset. Because these sorts of outliers are frequently found after model deployment, the model must be capable of detecting them while running to avoid performance degradation. A novel generative deep learning model based on Generative Adversarial Networks (GAN) and Variational AutoEncoders (VAE) that differentiates inliers from outliers using uniform distributions generated by variational autoencoders has been used in this report. Using the generative and adversarial elements of the model, three major losses (i.e., reconstruction loss, KL-divergence, and discriminative loss) are generated. As a result, we compare Alphagan to more known algorithms like Kmeans, LOF, OC-SVM, and DBSCAN. We demonstrated that first most efficient method is Alphagan an F1-score of 67.8% and an accuracy of 96%. Out of 9157 Questions/Answers 8803 were correctly predicted. And out of 706 positive Q/A, 352 were predicted as outliers.

We also designed a pipeline for the extraction of answers. We used Haystack as the platform for massive document collections to create end-to-end question answering systems.

Among various, Haystack offers significant advantages in document retrieval, as well as integration of pretrained models from Huggingface. Haystack provides a comprehensive framework for extracting information from Elasticsearch, which aids in retrieving context for BERT-based QA.

We then use the RoITD dataset to evaluate on various models such as bert-base-multilingual-uncased (mBERT), bert-base-romanian-uncased-v1 (romBERT), distilbert-base-multilingual-uncased (distilBERT), and ALR-BERT-cased (Romanian ALBERT). We employed two evaluation metrics (i.e., EM and F1-score) to assess MRC model performance on our dataset, consistent with evaluations on English SQuAD. It was observed that most of the models performs almost similar for EM. However, there was significant difference in F1-Score. We also made comparison for execution of each answer for the selected baseline models. We observed that execution time of all selected algorithms are not significantly distinct based on search algorithms but showed huge impact based on the baseline models. In all brevity, selection of models seems more important than the search algorithms for answer extraction.

In addition to this, we also implement the the response from the model into chatbot using DialogFlow. We used Pepper as the chatbot platform and implemented on Google Cloud Service. First we trained the model using the selected baselines and then hosted it using google cloud service (GCS). GCS is then interacted to google dialogflow in order to generate a chatbot interface. The dialogflow is then linked to Pepper where it fetches the response and display it to the user. The bottleneck in this approach was the time it takes to load the trained bert model into the docker. Since, dialogflow only waits for 5 seconds, the response is empty. Hence, to tackle this problem we ran bert model in the docker continuously and fetched the response which reduced the response time below 3 seconds. In addition to chatbot interface, it was also implemented in web interface as demonstrate in this project.

We extend the QA system using Bayesian Inference. It uses next best answer using bayesian inference. We verified our hypothesis that suppressing the top prediction and updating the probability of the remaining tokens would drive the model to explore additional options, which would likely increase the model's recall. We evaluated our approach, we applied softmax function on the model's output logits. The highest probability is replaced by a 0 and the bayesian update step is performed. Following the prediction of the start and end indices, the correct and predicted responses are run through a string normalization

procedure. Two answers may indicate the same thing but are not an exact match. Because it is less probable that the projected response would match the true answer character for character, we abstract the answer into a set of tokens and use the intersection of the two sets of tokens to evaluate performance.

At last we explored the knowledge graphs for Question answering. Despite significant advancements in QA processes over the last decade, the most accurate current approaches use a mix of the language model (LM) and knowledge graph (KG). To integrate these techniques, the model must be capable of identifying information from a big KG and then performing joint reasoning between the QA context and KG structure. QA-GNN is an end-to-end hybrid LM-KG model created particularly for QA tasks, in which the LM encodes the QA context before retrieving KG subgraphs. QA-GNN performs QA tasks by combining relevance scoring and joint reasoning. Relevance score is performed by assessing the significance of each KG node in relation to the QA query provided. Joint reasoning entails connecting the question context to the KG in order to generate a joint graph that merges these two modalities into a single working graph. Hence, we integrate KG into our QA framework. In order to do so, there was a necessity to perform a semantic analysis on the text that extracts the fundamental structure composed of Subject, Predicate and Object. Here subject and object constituted nodes and the predicate constituted edges. In our case computed the embeddings for nodes and edges. We used series of tuples containing (context, question, answer), trained on Encoder-Decoder Model with attention using categorical cross-entropy loss function. We then used smaller dataset and trained again but with anchor loss function. Experiments and results showed that the model struggles with the dataset since the implementation of the anchor loss strategy presents some difficulties in achieving high performance.

7.2 Next steps

Question-answer (QA) is a rapidly developing domain. We have identified several new approaches that will work towards QA domain advancement. Of these, entailment trees, extra-linguistic contexts, and neural natural logic inference have been identified as the most promising. In this section, we will introduce each of these techniques, along with areas of future work.

7.2.1 Entailment Trees

Entailment trees rely on known facts from intermediate conclusions to form multi-premise entailment steps to form a tree that identifies the question and appropriate answer [57]. By clearly defining the chain of reasoning between the input question and chosen answer, entailment trees offer a potential solution to the missing provided rationale in other QA models. Advantages to providing a clear pathway between the QA pair include easier error identification or potential advancement of interactive machine learning techniques. Figure 7.1 provides an example of a multi-step entailment tree, which is comprised of distinct, multi-faceted textual entailment steps [55,57,120]. In support of this QA method, Dalvi et. al. have compiled a multi-step entailment dataset known as ENTAILMENTBANK [57]. ENTAILMENTBANK is comprised of 1,840 QA pairs along with their corresponding multi-step entailment trees and includes both large and small multi-step entailment problems. The entailment trees are formed using advanced annotators and contain an average of 2.7 entailment steps and 6.6 nodes.

Three primary explanation tasks are defined using ENTAILMENTBANK where a valid entailment tree is generated for a provided QA pair when given—(1) all relevant sentences, which are defined as the leaves of the entailment tree, (2) all relevant sentences along with some distractor sentences, or (3) a complete corpus [57]. The purpose of the entailment tree is to separate the generated derivation (i.e., line of reasoning), which illustrates the way evidence points to the answer, from the pragmatics of deciding which portions of the derivation should be provided to the user [57]. In other words, the correctness of the derivation is separated from the utility to allow for derivations to be evaluated more objectively. Preliminary entailment tree QA results using ENTAILMENTBANK successfully demonstrate the generation of reasonable trees [57]. When the model input included all necessary

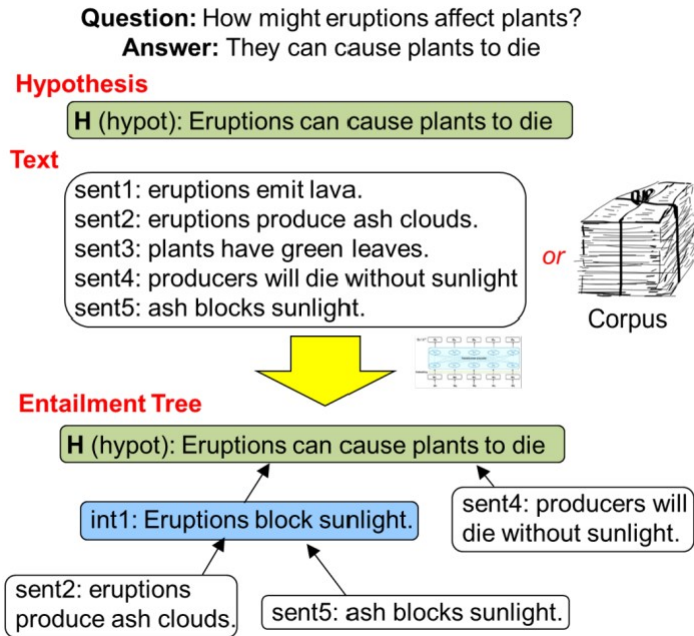


Figure 7.1: Entailment tree formation process. The given hypothesis (green), which summarizes the QA pair, is combined with a corpus or relevant text to create an entailment tree. The entailment tree includes intermediate nodes (blue) to illustrate how the hypothesis is connected to the corpus. [57]

raw facts, 35% of trees contained zero errors. Additionally, it is suggested that models trained using ENTAILMENTBANK may be generalized to other domains. Figure 7.2 provides an example of two medium-complexity entailment trees generated using this process, which takes approximately 20 minutes for each question [57].

Additional work is needed to develop this technique for advanced applications, although initial studies have demonstrated this as a technique with substantial promise. Error analysis of 100 entailment steps randomly sampled from imperfect entailment trees revealed that 30% were correct and 13% were nearly correct [57]. This indicated that invalid trees can still contain good steps. Additionally, invalid individual steps were caused by repetition, invalid entailment, and mis-evaluation and irrelevance. In the case of repetition, the derived conclusion repeats a portion of the input (41%) [57]. This has been attributed to high word overlap between the intermediate conclusions and input and may be improved by modifying the loss function so that the model generates unique conclusion statements compared with the input. In the case of invalid entailment, the model uses knowledge that is not provided in the input but is instead present elsewhere in the input context so that the conclusion does not follow from the input (47%) [57]. A correction for this may lie in the

implementation of a more interactive approach where entailment steps are generated individually before moving onto the next step. Finally, mis-evaluation and irrelevance errors occur when the conclusion is correct but differs from gold or cannot prove the hypothesis (12%) [57]. A possible correction method lies in improvement of the evaluation metric including modification of the loss function to include a goal-directed term that encourages intermediates closer to H . Other errors found in imperfect trees include incorrect or missing leaves (50%), imperfect evaluation (25%), correct leaves with invalid steps (20%), disconnected trees (5%), and correct steps with incorrect intermediate conclusions (<5%) [57].

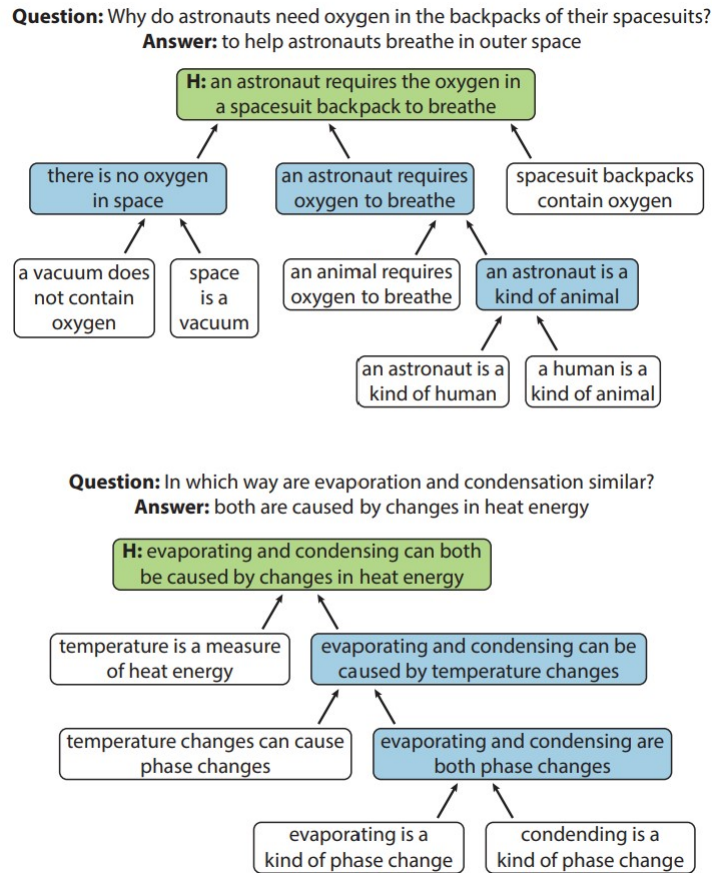


Figure 7.2: Two medium-complexity entailment tree examples. The hypotheses are the root nodes, identified as H (green). Intermediate conclusions are shown in blue. [57]

7.2.2 Extra-Linguistic Contexts

Questions may have different answers depending on the extra-linguistic contexts of when and where the question is asked. This often means that the answers to questions with extra-linguistic contexts can change over time, requiring consistent updating with the

most relevant information. Examples of these types of questions are illustrated in Figure 7.3. Questions with extra-linguistic contexts form a significant fraction of QA pairs, for instance approximately 16.5% of NQ-Open [231]. Zhang and Choi recently introduced an open-retrieval QA dataset named SITUATEDQA to incorporate temporal and geographical context when generating answers to input questions [231]. SITUATEDQA is comprised of temporal or geographical-dependent questions that are already present in existing QA datasets. Alternative contexts and their corresponding responses are identified via crowd-sourcing. Incorporating extra-linguistic contexts into language models allows for consistent updating of the corpus to maintain relevancy of generated answers.

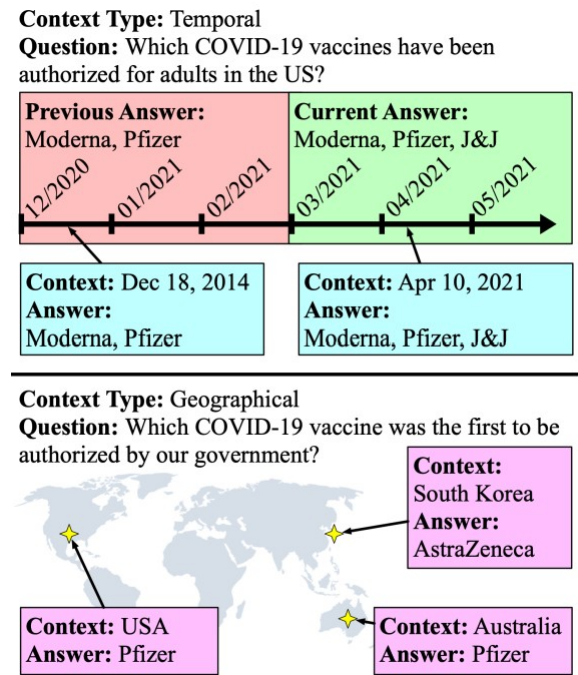


Figure 7.3: Sample questions with variable answers that depend on the geographical or temporal context. [231].

Current available QA systems do not consider extra-linguistic contexts. To account for extra-linguistic contexts in QA models, two tasks should be incorporated into the model— (1) what facts change over various extra-linguistic contexts and (2) how these facts change [231]. Other potential tasks include the consideration of who is asking the question, which would allow for individual preferences to be accounted for. In addition to maintaining corpus relevancy, these tasks can also be used to address errors caused by ambiguity.

7.2.3 Neural Natural Logic Inference

Neural natural logic inference works to improve QA models using a neural-symbolic QA approach as a means of integrating natural logic reasoning into deep learning architectures [188]. This allows for the model to gradually bridge a hypothesis and candidate premise by following natural logic inference steps to construct proof paths [188]. To decide whether a premise entails a given hypothesis, entailment scores between intermediate hypothesis and the candidate premise are calculated. This natural logic reasoning method produces a tree-like hierarchical structure. As a result, the hypotheses and premise is embedded in hyperbolic space instead of euclidean space. This allows for more precise representations to be acquired.

| Relation | Name | Example |
|-------------------|--------------------|------------------------------|
| $x \equiv y$ | equivalence | garbage \equiv rubbish |
| $x \sqsubseteq y$ | forward entailment | dog \sqsubseteq animal |
| $x \sqsupseteq y$ | reverse entailment | animal \sqsupseteq dog |
| $x \wedge y$ | negation | usual \wedge unusual |
| $x \Downarrow y$ | alternation | monkey \Downarrow elephant |
| $x \smile y$ | cover | mammal \smile nonhuman |
| $x \# y$ | independence | angry $\#$ fridge |

Figure 7.4: Summary of seven natural logic relations. [136, 188]

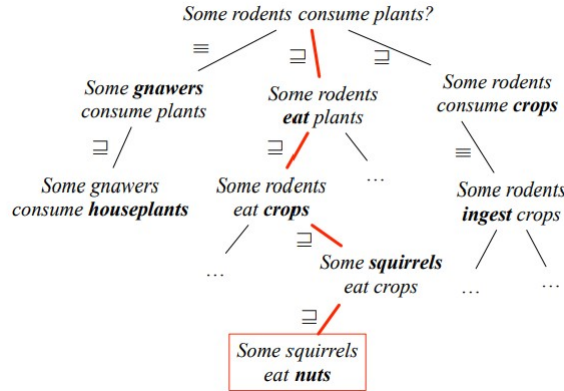


Figure 7.5: Example of the natural logic proof process, which begins with the provided hypothesis "rodents consume plants" to determine the premise "squirrels eat nuts". Labels located at the node edges define the logical relationships between the associated sentences. [188]

Natural logic proving occurs by the insertion, deletion, or mutation of words that follow monotonicity calculus or projectivity [188]. Seven primary logic relationships are summarized in Figure 7.4 [188]. Recently, Shi et. al. introduced the Neural Natural Logic

Inference (NeuNLI) QA framework to introduce neural natural logic inference [188]. This framework uses natural logic inference steps to bridge a given hypothesis and candidate premise by forming proof paths. This process starts by converting a question and associated potential answers to create a hypothesis in the form of declarative sentences. The original hypotheses are then rewritten to determine intermediate hypotheses. This process is repeated until a proof tree has been created for each QA pair. An example of the process is shown in Figure 7.5. The use of reasoning improves overall performance of the QA model compared with models that do not incorporate reasoning.

BIBLIOGRAPHY

- [1] Albert: A lite bert for self-supervised learning of language representations. <https://ai.googleblog.com/2019/12/albert-lite-bert-for-self-supervised.html>. Accessed: 2021-02-20.
- [2] Haystack. <https://github.com/deepset-ai/haystack/Haystack>. Accessed: 2022-02-10.
- [3] Kaldi CNN-TDNN LibriSpeech implementation.
- [4] Kaldi TDNN LibriSpeech implementation.
- [5] MS Windows NT kernel description. <https://haystack.deepset.ai/docs/intromd>. Accessed: 2021-02-10.
- [6] MS Windows NT kernel description. https://haystack.deepset.ai/docs/latest/use_casesmd. Accessed: 2020-02-11.
- [7] NetworkX. <https://networkx.org/NetworkX>. Accessed: 2022-02-10.
- [8] Nvidia OpenSeq2Seq Jasper LibriSpeech implementation.
- [9] Nvidia QuartzNet implementation.
- [10] PaddlePaddle DeepSpeech2 LibriSpeech implementation.
- [11] RWTH Returnn LibriSpeech implementation.
- [12] Spacy. <https://spacy.io>. Accessed: 2021-02-10.
- [13] Wav2Letter CNN-GLU fully convolutional LibriSpeech implementation.
- [14] Wav2Letter time-domain separable LibriSpeech implementation.
- [15] *A simple introduction to the KLT (Karhunen—Loève Transform)*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 151–179.
- [16] Kaldi Help Google Group: CNN-TDNN vs. TDNN, 2019.
- [17] Kaldi Help Google Group: Multiple output heads in chain network, 2019.

- [18] Open Speech and Language Resources, 2019.
- [19] Wav2Letter lexicon-free LibriSpeech implementation, 2019.
- [20] ABDEL-HAMID, O., MOHAMED, A.-R., JIANG, H., DENG, L., PENN, G., AND YU, D. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing* 22, 10 (2014), 1533–1545.
- [21] AFANASYEV, D. O., AND FEDOROVA, E. A. On the impact of outlier filtering on the electricity price forecasting accuracy. *Applied energy* 236 (2019), 196–210.
- [22] AKBIK, A., BERGMANN, T., BLYTHE, D., RASUL, K., SCHWETER, S., AND VOLLGRAF, R. FLAIR: An easy-to-use framework for state-of-the-art NLP. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics* (Minneapolis, Minnesota, 2019), Association for Computational Linguistics, pp. 54–59.
- [23] ALAM, M. Support vector machine (svm) for anomaly detection, 2022.
- [24] ALAM, M., SAMAD, M. D., VIDYARANTNE, L., GLANDON, A., AND IFTEKHARUDDIN, K. M. Survey on deep neural networks in speech and vision systems. *Neurocomputing* 417 (2020), 302–321.
- [25] ALQIFARI, R. Question answering systems approaches and challenges. In *Proceedings of the Student Research Workshop Associated with RANLP 2019* (2019), pp. 69–75.
- [26] ALSABTI, K., RANKA, S., AND SINGH, V. An efficient k-means clustering algorithm.
- [27] AVILA, C. V. S., FRANCO, W., MAIA, J. G. R., AND VIDAL, V. Conquest: A framework for building template-based iqa chatbots for enterprise knowledge graphs. *Natural Language Processing and Information Systems* 12089 (2020), 60 – 72.
- [28] BAHDANAU, D., CHO, K., AND BENGIO, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).
- [29] BAHDANAU, D., CHO, K., AND BENGIO, Y. Neural machine translation by jointly learning to align and translate. *CoRR abs/1409.0473* (2015).

- [30] BAO, J., DUAN, N., YAN, Z., ZHOU, M., AND ZHAO, T. Constraint-based question answering with knowledge graph. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers* (2016), pp. 2503–2514.
- [31] BAST, H., AND HAUSSMANN, E. More accurate question answering on freebase. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management* (2015), pp. 1431–1440.
- [32] BAUM, L. E., PETRIE, T., SOULES, G., AND WEISS, N. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics* 41, 1 (1970), 164–171.
- [33] BEN-GAL, I. Outlier detection. In *Data mining and knowledge discovery handbook*. Springer, 2005, pp. 131–146.
- [34] BERANT, J., CHOU, A., FROSTIG, R., AND LIANG, P. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing* (2013), pp. 1533–1544.
- [35] BERGER, J. The case for objective bayesian analysis. *Bayesian analysis* 1, 3 (2006), 385–402.
- [36] BERTINI, M., DEL BIMBO, A., AND SEIDENARI, L. Multi-scale and real-time non-parametric approach for anomaly detection and localization. *Computer Vision and Image Understanding* 116, 3 (2012), 320–329.
- [37] BHATTARAI, B., YADAV, R. K., GANG, H.-S., AND PYUN, J.-Y. Geomagnetic field based indoor landmark classification using deep learning. *IEEE Access* 7 (2019), 33943–33956.
- [38] BISWAS, A., YILMAZ, E., DE WET, F., VAN DER WESTHUIZEN, E., AND NIESLER, T. Semi-supervised acoustic model training for five-lingual code-switched asr. *arXiv preprint arXiv:1906.08647* (2019).
- [39] BOLLACKER, K., EVANS, C., PARITOSH, P., STURGE, T., AND TAYLOR, J. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1247–1250.

- [40] BORDES, A., USUNIER, N., GARCIA-DURAN, A., WESTON, J., AND YAKHNENKO, O. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems* 26 (2013).
- [41] BOSSELUT, A., RASHKIN, H., SAP, M., MALAVIYA, C., CELIKYILMAZ, A., AND CHOI, Y. Comet: Commonsense transformers for automatic knowledge graph construction. *arXiv preprint arXiv:1906.05317* (2019).
- [42] BROWN, L. D. In-season prediction of batting averages: A field test of empirical bayes and bayes methodologies. *The Annals of Applied Statistics* 2, 1 (2008), 113–152.
- [43] CANDEL, M. J., AND WINKENS, B. Performance of empirical bayes estimators of level-2 random parameters in multilevel analysis: A monte carlo study for longitudinal designs. *Journal of Educational and Behavioral Statistics* 28, 2 (2003), 169–194.
- [44] CHAN, W., JAITLEY, N., LE, Q., AND VINYALS, O. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *ICASSP* (2016), pp. 4960–4964.
- [45] CHEN, D., FISCH, A., WESTON, J., AND BORDES, A. Reading wikipedia to answer open-domain questions. In *ACL* (2017).
- [46] CHEN, Y., ZHOU, X. S., AND HUANG, T. S. One-class svm for learning in image retrieval. In *Proceedings 2001 International Conference on Image Processing (Cat. No. 01CH37205)* (2001), vol. 1, IEEE, pp. 34–37.
- [47] CHIU, C.-C., SAINATH, T. N., WU, Y., PRABHAVALKAR, R., NGUYEN, P., CHEN, Z., KANNAN, A., WEIS, R. J., RAO, K., AND GONINA, E. State-of-the-art speech recognition with sequence-to-sequence models. In *ICASSP* (2018), pp. 4774–4778.
- [48] CHO, K., VAN MERRIENBOER, B., BAHDANAU, D., AND BENGIO, Y. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).
- [49] CHOMET, H., PLESNIK, S., NICOLAE, C., DUNHAM, J., GOVER, L., WEAVING, T., AND FARIA, C. F. D. M. Controlling quantum effects in enhanced strong-field ionisation with machine-learning techniques, 2022.

- [50] CIERI, C., MILLER, D., AND WALKER, K. The fisher corpus: a resource for the next generations of speech-to-text. In *LREC* (2004), pp. 69–71.
- [51] COLLOBERT, R., HANNUN, A., AND SYNNAEVE, G. Word-level speech recognition with a dynamic lexicon. *arXiv preprint arXiv:1906.04323* (2019).
- [52] COLLOBERT, R., PUHRSCHE, C., AND SYNNAEVE, G. Wav2letter: an end-to-end convnet-based speech recognition system. *arXiv preprint arXiv:1609.03193* (2016).
- [53] CONNEAU, A., KHANDLWAL, K., GOYAL, N., CHAUDHARY, V., WENZKE, G., GUZMÁN, F., GRAVE, E., OTT, M., ZETTMAYER, L., AND STOYANOV, V. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (Online, July 2020), Association for Computational Linguistics, pp. 8440–8451.
- [54] CUI, Y., LIU, T., CHE, W., XIAO, L., CHEN, Z., MA, W., WANG, S., AND HU, G. A span-extraction dataset for Chinese machine reading comprehension. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (Hong Kong, China, Nov. 2019), Association for Computational Linguistics, pp. 5883–5889.
- [55] DAGAN, I., ROTH, D., SAMMONS, M., AND ZANZOTTO, F. M. Recognizing textual entailment: Models and applications. *Synthesis Lectures on Human Language Technologies* 6, 4 (2013), 1–220.
- [56] DAI, Z., YANG, Z., YANG, Y., COHEN, W. W., CARBONELL, J., LE, Q. V., AND SALAKHUTDINOV, R. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860* (2019).
- [57] DALVI, B., JANSEN, P., TAFJORD, O., XIE, Z., SMITH, H., PIPATANANGKURA, L., AND CLARK, P. Explaining answers with entailment trees. *arXiv preprint arXiv:2104.08661* (2021).
- [58] DARNIEDER, W. F. *Bayesian methods for data-dependent priors*. Thesis, 2011.
- [59] DAS, R., ZAHEER, M., REDDY, S., AND MCCALLUM, A. Question answering on knowledge bases and text using universal schema and memory networks. *arXiv preprint arXiv:1704.08384* (2017).

- [60] DAUPHIN, Y. N., FAN, A., AULI, M., AND GRANGIER, D. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (2017), pp. 933–941.
- [61] DAVIS, S., AND MERMELSTEIN, P. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE transactions on acoustics, speech, and signal processing* 28, 4 (1980), 357–366.
- [62] DEHAK, N., KENNY, P. J., DEHAK, R., DUMOUCHEL, P., AND OUELLET, P. Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing* 19, 4 (2010), 788–798.
- [63] DENG, L. M., AND YU, D. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing* 7 (2014), 197–387.
- [64] DEVLIN, J., CHANG, M.-W., LEE, K., AND TOUTANOVA, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv abs/1810.04805* (2019).
- [65] DEVLIN, J., CHANG, M.-W., LEE, K., AND TOUTANOVA, K. BERT: Pre-training of deep bidirectional transformers for language understanding. 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)* (Minneapolis, Minnesota, June 2019), Association for Computational Linguistics, pp. 4171–4186.
- [66] DHINGRA, B., MAZAITIS, K., AND COHEN, W. W. Quasar: Datasets for question answering by search and reading. *ArXiv abs/1707.03904* (2017).
- [67] D’HOFFSCHMIDT, M., BELBLIDIA, W., HEINRICH, Q., BRENDLÉ, T., AND VIDAL, M. FQuAD: French question answering dataset. In *Findings of the Association for Computational Linguistics: EMNLP 2020* (Online, Nov. 2020), Association for Computational Linguistics, pp. 1193–1208.
- [68] DOSHI-VELEZ, F., AND KIM, B. Towards a rigorous science of interpretable machine learning. *arXiv: Machine Learning* (2017).
- [69] DUBEY, M., BANERJEE, D., ABDELKAWI, A., AND LEHMANN, J. Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia. In *International semantic web conference* (2019), Springer, pp. 69–78.

- [70] EYHARABIDE, V., BEKKOUCH, I. E. I., AND CONSTANTIN, N. D. Knowledge graph embedding-based domain adaptation for musical instrument recognition. *Computers* 10, 8 (2021), 94.
- [71] EYHARABIDE, V., LULLY, V., AND MOREL, F. Musickg: Representations of sound and music in the middle ages as linked open data. In *International Conference on Semantic Systems* (2019), Springer, pp. 57–63.
- [72] FAN, H., JIANG, M., XU, L., ZHU, H., CHENG, J., AND JIANG, J. Comparison of long short term memory networks and the hydrological model in runoff simulation. *Water* 12 (2020), 175.
- [73] FENG, M., XIANG, B., GLASS, M. R., WANG, L., AND ZHOU, B. Applying deep learning to answer selection: A study and an open task. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)* (2015), IEEE, pp. 813–820.
- [74] FENG, Y., CHEN, X., LIN, B. Y., WANG, P., YAN, J., AND REN, X. Scalable multi-hop relational reasoning for knowledge-aware question answering. *arXiv preprint arXiv:2005.00646* (2020).
- [75] FORERO, M. G., CRISTÓBAL, G., AND DESCO, M. Automatic identification of mycobacterium tuberculosis by gaussian mixture models. *Journal of microscopy* 223, 2 (2006), 120–132.
- [76] FU, B., QIU, Y., TANG, C., LI, Y., YU, H., AND SUN, J. A survey on complex question answering over knowledge base: Recent advances and challenges. *arXiv preprint arXiv:2007.13069* (2020).
- [77] FUNG, W.-K., ZHU, Z.-Y., WEI, B.-C., AND HE, X. Influence diagnostics and outlier tests for semiparametric mixed models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 64, 3 (2002), 565–579.
- [78] GAL, Y., AND GHAHRAMANI, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, PMLR, pp. 1050–1059.
- [79] GAL, Y., AND GHAHRAMANI, Z. Bayesian convolutional neural networks with bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158* (2015).

- [80] GARDNER, M., GRUS, J., NEUMANN, M., TAFJORD, O., DASIGI, P., LIU, N., PETERS, M., SCHMITZ, M., AND ZETTLEMOYER, L. Allennlp: A deep semantic natural language processing platform, 2018.
- [81] GEORGESCU, A.-L., PAPPALARDO, A., CUCU, H., AND BLOTT, M. Performance vs . Hardware Requirements in State-Of-The-Art Automatic Speech Recognition. *IEEE/ACM TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING*, 1–25.
- [82] GIDIOTIS, A., AND TSOUMAKAS, G. Bayesian active summarization. *arXiv preprint arXiv:2110.04480* (2021).
- [83] GINSBURG, B., CASTONGUAY, P., HRINCHUK, O., KUCHAIEV, O., LAVRUKHIN, V., LEARY, R., LI, J., NGUYEN, H., AND COHEN, J. M. Stochastic gradient methods with layer-wise adaptive moments for training of deep networks. *arXiv preprint arXiv:1905.11286* (2019).
- [84] GODFREY, J. J., HOLLIMAN, E. C., AND MCDANIEL, J. Switchboard: Telephone speech corpus for research and development. In *ICASSP* (1992), pp. 517–520.
- [85] GOODMAN, J. A bit of progress in language modeling. *ArXiv cs.CL/0108005* (2001).
- [86] GRAVES, A., FERNANDEZ, S., GOMEZ, F., AND SCHMIDHUBER, J. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks,”. In *Proceedings of the 23rd international conference on Machine learning* (2006), pp. 369–376.
- [87] GRAVES, A., MOHAMED, A.-R., AND HINTON, G. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing* (2013), pp. 6645–6649.
- [88] GREEN, B. F., WOLF, A. K., CHOMSKY, C. L., AND LAUGHERY, K. Baseball: an automatic question-answerer. In *IRE-AIEE-ACM '61 (Western)* (1961).
- [89] GREEN JR, B. F., WOLF, A. K., CHOMSKY, C., AND LAUGHERY, K. Baseball: an automatic question-answerer. In *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference* (1961), pp. 219–224.

- [90] GULCEHRE, C., FIRAT, O., XU, K., CHO, K., BARRAULT, L., LIN, H.-C., BOUGARES, F., SCHWENK, H., AND BENGIO, Y. On using monolingual corpora in neural machine translation. *arXiv preprint arXiv:1503.03535* (2015).
- [91] GUU, K., MILLER, J., AND LIANG, P. Traversing knowledge graphs in vector space. *arXiv preprint arXiv:1506.01094* (2015).
- [92] HADIAN, H., SAMETI, H., POVEY, D., AND KHUDANPUR, S. Interspeech. *Inter-speech* (2018), 12–16.
- [93] HAMMING, R. W. Digital filters. *Courier Corporation* (1998).
- [94] HAN, S., KANG, J., MAO, H., HU, Y., LI, X., LI, Y., XIE, D., LUO, H., YAO, S., AND WANT, Y. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (2017), pp. 75–84.
- [95] HANNUN, A., LEE, A., XU, Q., AND COLLOBERT, R. Sequence-to-Sequence Speech Recognition with Time-Depth Separable Convolutions. In *Proc. Interspeech 2019* (2019), pp. 3785–3789.
- [96] HAO, Y., ZHANG, Y., LIU, K., HE, S., LIU, Z., WU, H., AND ZHAO, J. An end-to-end model for question answering over knowledge base with cross-attention combining global knowledge. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2017), pp. 221–231.
- [97] HAWKINS, D. M. *Identification of outliers*, vol. 11. Springer, 1980.
- [98] HE, W., LIU, K., LIU, J., LYU, Y., ZHAO, S., XIAO, X., LIU, Y., WANG, Y., WU, H., SHE, Q., LIU, X., WU, T., AND WANG, H. DuReader: a Chinese machine reading comprehension dataset from real-world applications. In *Proceedings of the Workshop on Machine Reading for Question Answering* (Melbourne, Australia, July 2018), Association for Computational Linguistics, pp. 37–46.
- [99] HERMANN, K. M., KOČISKÝ, T., GREFFENSTETTE, E., ESPEHOLT, L., KAY, W., SULEYMAN, M., AND BLUNSOM, P. Teaching machines to read and comprehend. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1* (Cambridge, MA, USA, 2015), NIPS’15, MIT Press, p. 1693–1701.

- [100] HILL, F., BORDES, A., CHOPRA, S., AND WESTON, J. The goldilocks principle: Reading children’s books with explicit memory representations. *arXiv preprint arXiv:1511.02301* (2015).
- [101] HINTON, G., DENG, L., YU, D., DAHL, G., MOHAMED, A.-R., JAITLY, N., SENIOR, A., VANHOUCKE, V., NGUYEN, P., AND KINGSBURY, B. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine* 29 (2012).
- [102] HONNIBAL, M., AND MONTANI, I. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear* (2017).
- [103] HORI, T., WATANABE, S., ZHANG, Y., AND CHAN, W. Advances in joint ct-cattention based end-to-end speech recognition with a deep cnn encoder and rnn-lm. *arXiv preprint arXiv:1706.02737* (2017).
- [104] HOVY, E., GERBER, L., HERMJAKOB, U., LIN, C.-Y., AND RAVICHANDRAN, D. Toward semantics-based answer pinpointing. In *Proceedings of the First International Conference on Human Language Technology Research* (2001).
- [105] HOWARD, G. S., MAXWELL, S. E., AND FLEMING, K. J. The proof of the pudding: an illustration of the relative strengths of null hypothesis, meta-analysis, and bayesian analysis. *Psychological methods* 5, 3 (2000), 315.
- [106] HYVÖNEN, E., ET AL. “sampo” model and semantic portals for digital humanities on the semantic web. In *Proceedings of the Digital Humanities in the Nordic Countries 5th Conference (DHN 2020)* (2020), CEUR-WS. org.
- [107] IBRAHIM, B. I., NICOLAE, D. C., KHAN, A., ALI, S. I., AND KHATTAK, A. Vae-gan based zero-shot outlier detection. In *Proceedings of the 2020 4th International Symposium on Computer Science and Intelligent Control* (2020), pp. 1–5.
- [108] IBRAHIM, J. G., CHEN, M., GWON, Y., AND CHEN, F. The power prior: theory and applications. *Statistics in medicine* 34, 28 (2015), 3724–3749.
- [109] JEDOUI, K., KRISHNA, R., BERNSTEIN, M., AND FEI-FEI, L. Deep bayesian active learning for multiple correct outputs. *arXiv preprint arXiv:1912.01119* (2019).

- [110] JOHNSON, A. L., AND MCGINNIS, L. F. Outlier detection in two-stage semi-parametric dea models. *European Journal of Operational Research* 187, 2 (2008), 629–635.
- [111] JOHNSON, S. R., TOMLINSON, G. A., HAWKER, G. A., GRANTON, J. T., AND FELDMAN, B. M. Methods to elicit beliefs for bayesian priors: a systematic review. *Journal of clinical epidemiology* 63, 4 (2010), 355–369.
- [112] JOSHI, M., CHOI, E., WELD, D. S., AND ZETTLEMOYER, L. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *ACL* (2017).
- [113] KANNAN, A., WU, Y., NGUYEN, P., SAINATH, T. N., CHEN, Z., AND PRABHAVALKAR, R. An analysis of incorporating an external language model into a sequence-to-sequence model. In *ICASSP* (2018), pp. 1–5828.
- [114] KASSNER, N., AND SCHÜTZE, H. Negated and misprimed probes for pretrained language models: Birds can talk, but cannot fly. *arXiv preprint arXiv:1911.03343* (2019).
- [115] KENNY, P., BOULIANNE, G., OUELLET, P., AND DUMOUCHEL, P. Joint factor analysis versus eigenchannels in speaker recognition. *IEEE Transactions on Audio, Speech, and Language Processing* 15, 4 (2007), 1435–1447.
- [116] KREYSSIG, F. L., ZHANG, C., AND WOODLAND, P. C. Improved tdnnns using deep kernels and frequency dependent grid-rnns. In *ICASSP* (2018), pp. 4864–4868.
- [117] KRIMAN, S., BELIAEV, S., GINSBURG, B., HUANG, J., KUCHAIEV, O., LAVRUKHIN, V., LEARY, R., LI, J., AND ZHANG, Y. Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions. In *ICASSP* (2020), pp. 6124–6128.
- [118] KULLBACK, S., AND LEIBLER, R. A. On information and sufficiency. *The annals of mathematical statistics* 22, 1 (1951), 79–86.
- [119] KWIATKOWSKI, T., PALOMAKI, J., REDFIELD, O., COLLINS, M., PARIKH, A., ALBERTI, C., EPSTEIN, D., POLOSUKHIN, I., KELCEY, M., DEVLIN, J., LEE, K., TOUTANOVA, K. N., JONES, L., CHANG, M.-W., DAI, A., USZKOREIT, J., LE, Q., AND PETROV, S. Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics* (2019).

- [120] LAI, A., BISK, Y., AND HOCKENMAIER, J. Natural language inference from multiple premises. *arXiv preprint arXiv:1710.02925* (2017).
- [121] LAI, G., XIE, Q., LIU, H., YANG, Y., AND HOVY, E. H. Race: Large-scale reading comprehension dataset from examinations. In *EMNLP* (2017).
- [122] LAN, Z., CHEN, M., GOODMAN, S., GIMPEL, K., SHARMA, P., AND SORICUT, R. ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR abs/1909.11942* (2019).
- [123] LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 11 (1998), 2278–2324.
- [124] LECUN, Y., HAFFNER, P., BOTTOU, L., AND BENGIO, Y. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision* (1999).
- [125] LEE, M., HWANG, K., PARK, J., CHOI, S., SHIN, S., AND SUNG, W. Fpga-based low-power speech recognition with recurrent neural networks. In *2016 IEEE International Workshop on Signal Processing Systems (SiPS)* (2016), pp. 230–235.
- [126] LI, J., CHEN, X., HOVY, E. H., AND JURAFSKY, D. Visualizing and understanding neural models in nlp. In *HLT-NAACL* (2016).
- [127] LI, X., AND ROTH, D. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics* (2002).
- [128] LIKHOMANENKO, T., SYNNAEVE, G., AND COLLOBERT, R. Who needs words? lexicon-free speech recognition. *arXiv preprint arXiv:1904.04479* (2019).
- [129] LIM, S., KIM, M., AND LEE, J. Korquad1.0: Korean qa dataset for machine reading comprehension, 2019.
- [130] LIN, B. Y., CHEN, X., CHEN, J., AND REN, X. Kagnet: Knowledge-aware graph networks for commonsense reasoning. *arXiv preprint arXiv:1909.02151* (2019).
- [131] LIPTON, Z. C. A critical review of recurrent neural networks for sequence learning. *ArXiv abs/1506.00019* (2015).

- [132] LIU, B., QUIN, H., GONG, Y., GE, W., XIA, M., AND SHI, L. Eera-asr: An energy-efficient reconfigurable architecture for automatic speech recognition with hybrid dnn and approximate computing. *IEEE Access* 6 (2018).
- [133] LOPEZ-COZAR, R. D., AND ARAKI, M. *Spoken, multilingual and multimodal dialogue systems: development and assessment*. 2005.
- [134] LUONG, T., PHAM, H., AND MANNING, C. D. Effective approaches to attention-based neural machine translation. In *EMNLP* (2015).
- [135] LV, S., GUO, D., XU, J., TANG, D., DUAN, N., GONG, M., SHOU, L., JIANG, D., CAO, G., AND HU, S. Graph-based reasoning over heterogeneous external knowledge for commonsense question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2020), vol. 34, pp. 8449–8456.
- [136] MACCARTNEY, B., AND MANNING, C. D. An extended model of natural logic. In *Proceedings of the eight international conference on computational semantics* (2009), pp. 140–156.
- [137] MAKLIN, C. Gaussian mixture models clustering algorithm explained, Jul 2019.
- [138] MASALA, M., RUSETI, S., AND DASCALU, M. RoBERT – a Romanian BERT model. In *Proceedings of the 28th International Conference on Computational Linguistics* (Barcelona, Spain (Online), Dec. 2020), International Committee on Computational Linguistics, pp. 6626–6637.
- [139] MEGHINI, C., BARTALESI, V., AND METILLI, D. Representing narratives in digital libraries: The narrative ontology. *Semantic Web* 12, 2 (2021), 241–264.
- [140] MIGACZ, S. 8-bit inference with tensorrt. In *GPU technology conference* (2017), p. 5.
- [141] MIKOLOV, T., KARAFIAT, M., BURGET, L., CERNOCKY, J., AND KHUDANPUR, S. Recurrent neural network based language model. In *Interspeech* (2010).
- [142] MIKOLOV, T., KOMBRINK, S., DEORAS, A., BURGET, L., AND CERNOCKY, J. Rnnlm-recurrent neural network language modeling toolkit. In *Proc. of the 2011 ASRU Workshop* (2011), pp. 196–201.

- [143] MISHRA, A., AND VISHWAKARMA, S. Analysis of tf-idf model and its variant for document retrieval. In *2015 International Conference on Computational Intelligence and Communication Networks (CICN)* (2015), pp. 772–776.
- [144] MOHAMADI, S., AND AMINDAVAR, H. Deep bayesian active learning, a brief survey on recent advances. *arXiv preprint arXiv:2012.08044* (2020).
- [145] MORRIS, D. E., OAKLEY, J. E., AND CROWE, J. A. A web-based tool for eliciting probability distributions from experts. *Environmental Modelling and Software* 52 (2014), 1–4.
- [146] NEAL, R. M. Pattern recognition and machine learning. *Technometrics* 49 (2007), 366.
- [147] NEWTON, M. A., NOUEIRY, A., SARKAR, D., AND AHLQUIST, P. Detecting differential gene expression with a semiparametric hierarchical mixture method. *Biostatistics* 5, 2 (2004), 155–176.
- [148] NICOLAE, C. Romanian it dataset (roitd).
- [149] NICOLAE, C., AND TUFIS, D. Roitd: Romanian it question answering dataset. *ConsiLR 2021* (2011).
- [150] NOORALAHZADEH, F., ØVRELID, L., AND LØNNING, J. T. Evaluation of domain-specific word embeddings using knowledge resources. In *LREC* (2018).
- [151] O’HAGAN, A., BUCK, C. E., DANESHKHAH, A., EISER, J. R., GARTHWAITE, P. H., JENKINSON, D. J., OAKLEY, J. E., AND RAKOW, T. Uncertain judgements: eliciting experts’ probabilities.
- [152] OPPENHEIM, A. V. *Discrete-time signal processing*. Pearson Education, India, 1999.
- [153] PAGE, L., BRIN, S., MOTWANI, R., AND WINOGRAD, T. The pagerank citation ranking : Bringing order to the web. In *WWW 1999* (1999).
- [154] PAN, S. J., AND YANG, Q. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering* 22 (2010), 1345–1359.
- [155] PANAYOTOV, V., CHEN, G., POVEY, D., AND KHUDANPUR, S. Librispeech: an asr corpus based on public domain audio books. In *ICASSP* (2015), pp. 5206–5210.

- [156] PARANJPE, D., RAMAKRISHNAN, G., AND SRINIVASAN, S. Passage scoring for question answering via bayesian inference on lexical relations. In *TREC*, Citeseer, pp. 305–210.
- [157] PAUL, D. B., AND BAKER, J. M. The design for the wall street journal-based csr corpus. In *Proceedings of the workshop on Speech and Natural Language* (1992), pp. 357–362.
- [158] PEDDINTI, V., POVEY, D., AND KHUDANPUR, S. A time delay neural network architecture for efficient modeling of long temporal contexts. In *Interspeech* (2015).
- [159] PETERS, M. E., NEUMANN, M., IYYER, M., GARDNER, M., CLARK, C., LEE, K., AND ZETTMLOYER, L. Deep contextualized word representations. *ArXiv abs/1802.05365* (2018).
- [160] PETRONI, F., ROCKTÄSCHEL, T., LEWIS, P., BAKHTIN, A., WU, Y., MILLER, A. H., AND RIEDEL, S. Language models as knowledge bases? *arXiv preprint arXiv:1909.01066* (2019).
- [161] PINSLER, R., GORDON, J., NALISNICK, E., AND HERNÁNDEZ-LOBATO, J. M. Bayesian batch active learning as sparse subset approximation. *Advances in neural information processing systems* 32 (2019), 6359–6370.
- [162] POVEY, D., CHENG, G., WANG, Y., LI, K., XU, H., YARMOHAMMADI, M., AND KHUDANPUR, S. Semi-orthogonal low-rank matrix factorization for deep neural networks. In *Interspeech* (2018), pp. 3743–3747.
- [163] POVEY, D., PEDDINTI, V., GALVEZ, D., GHahremani, P., MANOHAR, V., NA, X., WANG, Y., AND KHUDANPUR, S. Purely sequence-trained neural networks for asr based on lattice-free mmi. In *Interspeech* (2016), pp. 2751–2755.
- [164] PRICE, M., GLASS, J., AND CHANDRAKASAN, A. P. A low-power speech recognizer and voice activity detector using deep neural networks. *IEEE Journal of Solid-State Circuits* 53, 1 (2017), 66–75.
- [165] QI, P., ZHANG, Y., ZHANG, Y., BOLTON, J., AND MANNING, C. D. Stanza: A python natural language processing toolkit for many human languages, 2020.
- [166] RAJPURKAR, P., JIA, R., AND LIANG, P. Know what you don’t know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the*

Association for Computational Linguistics (Volume 2: Short Papers) (Melbourne, Australia, July 2018), Association for Computational Linguistics, pp. 784–789.

- [167] RAJPURKAR, P., ZHANG, J., LOPYREV, K., AND LIANG, P. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (Austin, Texas, Nov. 2016), Association for Computational Linguistics, pp. 2383–2392.
- [168] RAJPURKAR, P., ZHANG, J., LOPYREV, K., AND LIANG, P. Squad: 100,000+ questions for machine comprehension of text, 2016.
- [169] REDDY, S., CHEN, D., AND MANNING, C. D. CoQA: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics* 7 (Mar. 2019), 249–266.
- [170] REN, H., HU, W., AND LESKOVEC, J. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. *arXiv preprint arXiv:2002.05969* (2020).
- [171] REN, H., AND LESKOVEC, J. Beta embeddings for multi-hop logical reasoning in knowledge graphs. *arXiv preprint arXiv:2010.11465* (2020).
- [172] RIBEIRO, M. T., SINGH, S., AND GUESTRIN, C. "why should i trust you?": Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016).
- [173] RICHARDSON, S., AND GREEN, P. J. On bayesian analysis of mixtures with an unknown number of components (with discussion). *Journal of the Royal Statistical Society: series B (statistical methodology)* 59, 4 (1997), 731–792.
- [174] RIETBERGEN, C., KLUGKIST, I., JANSSEN, K. J., MOONS, K. G., AND HOIJTINK, H. J. Incorporation of historical data in the analysis of randomized therapeutic trials. *Contemporary Clinical Trials* 32, 6 (2011), 848–855.
- [175] ROSENBLATT, F. F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* 65 6 (1958), 386–408.
- [176] ROUSSEAU, A., DELEGLISE, P., AND ESTEVE, Y. Enhancing the ted-lium corpus with selected data for language modeling and more ted talks. In *LREC* (2014), pp. 3935–3939.

- [177] RUDER, S. Neural transfer learning for natural language processing.
- [178] RYU, P.-M., JANG, M.-G., AND KIM, H. Open domain question answering using wikipedia-based knowledge model. *Inf. Process. Manag.* 50 (2014), 683–692.
- [179] SAINATH, T. N., PRABHAVALKAR, R., KUMAR, S., LEE, S., KANNAN, A., RYBACH, D., SCHOGOL, V., NGUYEN, P., LI, B., AND WU, Y. No need for a lexicon? evaluating the value of the pronunciation lexica in end-to-end models. In *ICASSP* (2018), pp. 5859–5863.
- [180] SALIMANS, T., AND KINGMA, D. P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems* (2016), pp. 901–909.
- [181] SALTON, G., AND MCGILL, M. Introduction to modern information retrieval.
- [182] SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.* 3 (1959), 210–229.
- [183] SANH, V., DEBUT, L., CHAUMOND, J., AND WOLF, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.
- [184] SCHALK, R. Clariah: Enabling interoperability between humanities disciplines with ontologies. *Applications and Practices in Ontology Design, Extraction, and Reasoning* 49 (2020), 73.
- [185] SCHUBERT, E., SANDER, J., ESTER, M., KRIEGEL, H. P., AND XU, X. Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)* 42, 3 (2017), 1–21.
- [186] SENNRICH, R., HADDOW, B., AND BIRCH, A. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909* (2015).
- [187] SHAN, C., ZHANG, J., WANG, Y., AND XIE, L. Attention-based end-to-end speech recognition on voice search. In *ICASSP* (2018), pp. 4764–4768.
- [188] SHI, J., DING, X., DU, L., LIU, T., AND QIN, B. Neural natural logic inference for interpretable question answering. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing* (2021), pp. 3673–3684.

- [189] SIMPSON, E., GAO, Y., AND GUREVYCH, I. Interactive text ranking with bayesian optimization: A case study on community qa and summarization. *Transactions of the Association for Computational Linguistics* 8 (2020), 759–775.
- [190] SPEER, R., CHIN, J., AND HAVASI, C. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Thirty-first AAAI conference on artificial intelligence*.
- [191] SRIRAM, A., JUN, H., SATHEESH, S., AND COATES, A. Cold fusion: Training seq2seq models together with language models. *arXiv preprint arXiv:1708.06426* (2017).
- [192] STEVENS, S. S., VOLKMANN, J., AND NEWMAN, E. B. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America* 8, 3 (1937), 185–190.
- [193] SUKHBAAATAR, S., SZLAM, A., WESTON, J., AND FERGUS, R. End-to-end memory networks. In *NIPS* (2015).
- [194] SUN, H., DHINGRA, B., ZAHEER, M., MAZAITIS, K., SALAKHUTDINOV, R., AND COHEN, W. W. Open domain question answering using early fusion of knowledge bases and text. *arXiv preprint arXiv:1809.00782* (2018).
- [195] SUNDERMEYER, M., SCHLUTER, R., AND NEY, H. Lstm neural networks for language modeling. In *Interspeech* (2012).
- [196] SUTSKEVER, I., VINYALS, O., AND LE, Q. Sequence to sequence learning with neural networks. *Advances in NIPS* (2014).
- [197] SUTSKEVER, I., VINYALS, O., AND LE, Q. V. Sequence to sequence learning with neural networks. *ArXiv abs/1409.3215* (2014).
- [198] SUTSKEVER, I., VINYALS, O., AND LE, Q. V. Sequence to sequence learning with neural networks. *ArXiv abs/1409.3215* (2014).
- [199] TALMOR, A., AND BERANT, J. The web as a knowledge-base for answering complex questions. *arXiv preprint arXiv:1803.06643* (2018).
- [200] TAN, M., SANTOS, C. D., XIANG, B., AND ZHOU, B. Lstm-based deep learning models for non-factoid answer selection. *arXiv preprint arXiv:1511.04108* (2015).

- [201] TOSHWI, S., KANNAN, A., CHIU, C.-C., WU, Y., SAINATH, T. N., AND LIVESCU, K. A comparison of techniques for language model integration in encoder-decoder speech recognition. In *2018 IEEE Spoken Language Technology Workshop (SLT)* (2018), pp. 369–375.
- [202] TRIVEDI, P., MAHESHWARI, G., DUBEY, M., AND LEHMANN, J. Lc-quad: A corpus for complex question answering over knowledge graphs. In *International Semantic Web Conference* (2017), Springer, pp. 210–218.
- [203] VAKULENKO, S., AND SAVENKOV, V. Tableqa: Question answering on tabular data, 2017.
- [204] VAN AELST, S., AND ROUSSEEUW, P. Minimum volume ellipsoid. *Wiley Interdisciplinary Reviews: Computational Statistics* 1, 1 (2009), 71–82.
- [205] VAN DE SCHOOT, R., DEPAOLI, S., KING, R., KRAMER, B., MÄRTENS, K., TADESSE, M. G., VANNUCCI, M., GELMAN, A., VEEN, D., AND WILLEMSSEN, J. Bayesian statistics and modelling. *Nature Reviews Methods Primers* 1, 1 (2021), 1–26.
- [206] VAN DE SCHOOT, R., SIJBRANDIJ, M., DEPAOLI, S., WINTER, S. D., OLFF, M., AND VAN LOEY, N. E. Bayesian ptsd-trajectory analysis with informed priors based on a systematic literature search and expert elicitation. *Multivariate Behavioral Research* 53, 2 (2018), 267–291.
- [207] VAN DER LINDEN, W. J. Using response times for item selection in adaptive testing. *Journal of Educational and Behavioral Statistics* 33, 1 (2008), 5–20.
- [208] VASWANI, A., BENGIO, S., BREVDO, E., CHOLLET, F., GOMEZ, A. N., GOUWS, S., JONES, L., ŁUKASZ KAISER, KALCHBRENNER, N., PARMAR, N., SEPASSI, R., SHAZEER, N., AND USZKOREIT, J. Tensor2tensor for neural machine translation, 2018.
- [209] VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, L., AND POLOSUKHIN, I. Attention is all you need. *ArXiv abs/1706.03762* (2017).
- [210] VEEN, D., STOEL, D., ZONDERVAN-ZWIJNENBURG, M., AND VAN DE SCHOOT, R. Proposal for a five-step method to elicit expert judgment. *Frontiers in psychology* 8 (2017), 2110.

- [211] VITERBI, A. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory* 13, 2 (1967), 260–269.
- [212] WAIBEL, A., HANAZAWA, T., HINTON, G., SHIKANO, K., AND LANG, K. J. Phoneme recognition using time-delay neural networks. *IEEE transactions on acoustics, speech, and signal processing* 37 (1989), 328–339.
- [213] WANG, D., WANG, X., AND LV, S. An overview of end-to-end automatic speech recognition. *Symmetry* 11, 8 (2019), 1018.
- [214] WANG, X., KAPANIPATHI, P., MUSA, R., YU, M., TALAMADUPULA, K., ABDELAZIZ, I., CHANG, M., FOKOUE, A., MAKNI, B., MATTEI, N., ET AL. Improving natural language inference using external knowledge in the science questions domain. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), vol. 33, pp. 7208–7215.
- [215] WASSERMAN, L. Asymptotic inference for mixture models by using data-dependent priors. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 62, 1 (2000), 159–180.
- [216] WEIZENBAUM, J. Eliza—a computer program for the study of natural language communication between man and machine. *Commun. ACM* 9 (1966), 36–45.
- [217] XU, H. A pruned rnnlm lattice-rescoring algorithm for automatic speech recognition. In *ICASSP* (2018), pp. 5929–5933.
- [218] YANG, J., WARD, M. O., RUNDENSTEINER, E. A., AND HUANG, S. Visual hierarchical dimension reduction for exploration of high dimensional datasets. In *VisSym* (2003).
- [219] YASUNAGA, M., REN, H., BOSSELUT, A., LIANG, P., AND LESKOVEC, J. Qa-gnn: Reasoning with language models and knowledge graphs for question answering. *arXiv preprint arXiv:2104.06378* (2021).
- [220] YAZDANI, R., SEGURA, A., ARNAU, J.-M., AND GONZALEZ, A. An ultra low-power hardware accelerator for automatic speech recognition. In *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (2016), pp. 1–12.

- [221] YIH, S. W.-T., CHANG, M.-W., HE, X., AND GAO, J. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the Joint Conference of the 53rd Annual Meeting of the ACL and the 7th International Joint Conference on Natural Language Processing of the AFNLP* (2015).
- [222] YIH, W.-T., RICHARDSON, M., MEEK, C., CHANG, M.-W., AND SUH, J. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (2016), pp. 201–206.
- [223] YILDIRIM, S. K-means clustering explained, Mar 2020.
- [224] YIN, W., YU, M., XIANG, B., ZHOU, B., AND SCHÜTZE, H. Simple question answering by attentive convolutional neural network. *arXiv preprint arXiv:1606.03391* (2016).
- [225] YOSINSKI, J., CLUNE, J., BENGIO, Y., AND LIPSON, H. How transferable are features in deep neural networks? *ArXiv abs/1411.1792* (2014).
- [226] YOU, Y., LI, J., REDDI, S., HSEU, J., KUMAR, S., BHOJANAPALLI, S., SONG, X., DEMMEL, J., KEUTZER, K., AND HSIEH, C.-J. Large batch optimization for deep learning: Training bert in 76 minutes. In *International Conference on Learning Representations* (2020).
- [227] YOUNG, S. J., ODELL, J. J., AND WOODLAND, P. C. Tree-based state tying for high accuracy acoustic modelling. In *Proceedings of the workshop on Human Language Technology* (1994), pp. 307–312.
- [228] YOUNG, T., HAZARIKA, D., PORIA, S., AND CAMBRIA, E. Recent trends in deep learning based natural language processing [review article]. *IEEE Computational Intelligence Magazine* 13 (2018), 55–75.
- [229] ZEGHIDOUR, N., USUNIER, N., SYNNAEVE, G., COLLOBERT, R., AND DUPOUX, E. End-to-end speech recognition from the raw waveform. *arXiv preprint arXiv:1806.07098* (2018).
- [230] ZEYER, A., IRIE, K., SCHLUTER, R., AND NEY, H. Improved training of end-to-end attention models for speech recognition. In *Proc. Interspeech 2018* (2018), pp. 7–11.

- [231] ZHANG, M. J., AND CHOI, E. Situatedqa: Incorporating extra-linguistic contexts into qa. *arXiv preprint arXiv:2109.06157* (2021).
- [232] ZHANG, Y., ALDER, M., AND TOGNERI, R. Using gaussian mixture modeling in speech recognition. In *ICASSP* (1994), pp. 1–613.
- [233] ZHANG, Y., LIU, K., HE, S., JI, G., LIU, Z., WU, H., AND ZHAO, J. Question answering over knowledge base with neural attention combining global knowledge information. *arXiv preprint arXiv:1606.00979* (2016).
- [234] ZORILA, C., BOEDDEKER, C., DODDIPATLA, R., AND HAEB-UMBACH, R. An investigation into the effectiveness of enhancement in asr training and test for chime-5 dinner party transcription. *arXiv preprint arXiv:1909.12208* (2019).