Fall 2023

Spoken Language Processing and Modeling for Aviation Communications

Aaron Van De Brook
Embry-Riddle Aeronautical University, vandebra@my.erau.edu

SPOKEN LANGUAGE PROCESSING AND MODELING FOR AVIATION COMMUNICATIONS

by

Aaron Van De Brook

This thesis was prepared under the direction of the candidate's Thesis Committee Chair, Dr. Jianhua Liu, Professor, Daytona Beach Campus, and Thesis Committee Members Andrew Schneider, Professor, Daytona Beach Campus, and Dr. Prashant Shekhar, Professor, Daytona Beach Campus, and has been approved by the Thesis Committee. It was submitted to the Department of Electrical Engineering & Computer Science in partial fulfillment of the requirements for the degree of Master of Science in Electrical & Computer Engineering.

Thesis Review Committee:

Jianhua Liu Digitally signed by Jianhua Liu DN: cn=Jianhua Liu, c=US, email=liu620@erau.edu Date: 2024.01.09 15:14:17 -05'00'

Jianhua Liu, Ph.D.
Committee Chair

Prashant Shekhar Digitally signed by Prashant Shekhar Date: 2024.01.09 16:10:23 -05'00'

Prashant Shekhar, Ph.D.
Committee Member

Andrew Schneider Digitally signed by Andrew Schneider Date: 2024.01.10 08:47:00 -05'00'

Andrew Schneider
Committee Member

Jianhua Liu Digitally signed by Jianhua Liu DN: cn=Jianhua Liu, c=US, email=liu620@erau.edu Date: 2024.01.09 15:15:31 -05'00'

Jianhua Liu, Ph.D.
MS ECE Program Coordinator,
Department of Electrical Engineering & Computer Science

[illegible] 1/10/24

Massood Towhidnejad, Ph.D.
Department Chair,
Electrical Engineering & Computer Science

Seetha Raghavan Digitally signed by Seetha Raghavan Date: 2024.01.11 17:01:26 -05'00'

James Gregory, Ph.D.
Dean, College of Engineering

Bert J. Zarb 01.12.2024

Bert Zarb, D.B.A.
Vice Provost for Academic Affairs

Date

Contents

1 Abstract

With recent advances in machine learning and deep learning technologies and the creation of larger aviation-specific corpora, applying natural language processing technologies, especially those based on transformer neural networks, to aviation communications is becoming increasingly feasible. Previous work has focused on machine learning applications to natural language processing, such as N-grams and word lattices. This thesis experiments with a process for pretraining transformer-based language models on aviation English corpora and compare the effectiveness and performance of language models transfer learned from pretrained checkpoints and those trained from their base weight initializations (trained from scratch). The results suggest that transformer language models trained from scratch outperform models fine-tuned from pretrained checkpoints. The work concludes by recommending future work to improve pretraining performance and suggestions for downstream, in-domain tasks such as semantic extraction, named entity recognition (callsign identification), speaker role identification, and speech recognition.

2 Introduction

There have been noticeable and significant advancements in the general domain of artificial intelligence (AI) recently. Natural language processing and language modeling technologies, in particular, have made tremendous advances over the past two decades [1–4], achieving performance levels good enough to be brought to consumer-facing products such as personal assistants, search engines, phones, televisions, etc. Due to the success in the general domain, applications have begun to spin off into problem-specific domains, such as (and most notably for this work) aviation. For example, several speech recognition systems with rule- and machine learning-based natural language processing, inference, and semantic extraction algorithms have been implemented at EUROCONTROL simulation centers in Europe to train air traffic controllers and study the potential effects of AI-augmented air traffic management (ATM) systems on controller workloads [5,6]. Preliminary studies have found that AI-augmented ATM systems reduce controller workloads and fuel consumption of aircraft by optimizing aircraft routes in the airspace and reducing departure/arrival times, thereby reducing fuel consumption [5].

At the time of this writing, the current focus of general machine learning and deep learning algorithms in the language modeling domains has been on a variety of deep learning methods/architectures, namely, recurrent, convolutional, and transformer neural networks, due to the availability of large natural language corpora such as WikiText [7] and IMDB [8]. In contrast to the general domain, the aviation domain still prefers and sees much success with traditional machine learning models for language modeling applications. Typically, the most prevalent of the traditional models are N-grams, word lattices, and rule-based approaches, depending on the application and operating environment of the models. The most likely reason for this divergence between the domains is the difference and relative lack of labeled data in the aviation domain. By comparison, the largest corpora in the general domain are either WikiText or the BookCorpus, which contain trillions of tokens, whereas, for aviation, the largest corpus is Air Traffic Control Complete with approximately 26 hours of labeled data amounting to just over 300,000 tokens[1]. This disparity in data availability is exacerbated by the fact that aviation English uses a specialized vocabulary in addition to specialized pronunciation for some words to enhance understandability in a radiotelephonic (R/T) medium [9]. This makes the data labeling process significantly more difficult because it not only requires two types of data (audio and text) to create labels, but also requires labelers to have domain knowledge and experience to create effective transcriptions. This, in addition to the following facets, further contributes to the difficulty of transcription: (1) the medium of communication, (2) the environment in which the communication occurs[2], (3) a noisy and active communication environment i.e. an active flight deck under the

[1]Air Traffic Control Complete has approximately 70 hours of audio data, however, about 26 hours of the 70 are labeled.

[2]For example, a noisy and possibly low fidelity medium, depending on the equipment used to transmit and receive communications

influence of wind, engine noise, etc., (4) an active Air Traffic Control (ATC) tower and channels contributing to increased background noise on the ATC side, and (5) a rapid rate of speech. Pilots are also operating sophisticated equipment during while communicating, requiring significant attention and mental resources further contributing to the complexity of the communications [9].

The potential for speech recognition and language modeling algorithms to reach better than human performance (for example, the Stanford Question Answering Dataset (SQuAD) benchmark has recorded results consistently better than humans since 2019[3]) [10] suggests that these NLP algorithms could have high utility in the aviation domain. If they can achieve better-than-human performance in the aviation domain and be used with pilot and air traffic controller systems, they could easily boost efficiency, reduce workloads, and reduce and mitigate errors during normal operations.

The methods and results laid out in this thesis were done with the intention of creating a streamlined process for training and testing deep learning-based NLP methods, specifically language modeling. The results from the language models suggest that the volume of text used to train the language models is not yet sufficient to produce results akin to those in the conversational and/or literary English domains. Additionally, the results in Section 6.6.6 show that language models trained "from scratch" on in-domain data perform better than models trained on out-of-domain data and transfer learned on in-domain data.

2.1 Problem Statement

Other work has shown that the word error rates of automatic speech recognition models can and have been significantly reduced by using language models to determine the most likely token at a time step in a sequence [3, 4]. Language models have also been shown to be effective for natural language processing tasks such as named entity recognition [11, 12]. Speech recognition and callsign detection (CSD) have become popular tasks for machine

[3]`https://rajpurkar.github.io/SQuAD-explorer/`

learning applications in aviation [5,13,14]; callsign detection is a task analogous to named entity recognition since the elements of a callsign in a transcription can be treated as named entities in a sentence. For example, in the following transcription

skyshuttle one one four zero now descend flight level three three zero

the sequence “skyshuttle one one four zero” should be detected as the callsign. This is similar to detecting “EU”, “German”, and “British” as named entities in the sequence below.

EU rejects German call to boycott British lamb

The key component missing for both of these applications is an effectively pretrained language model that can be used for these downstream tasks, so the goal of this thesis is to run several experiments using transformer-based language models to determine the most effective strategy for pretraining the model for downstream tasks. Recommended best practice is to use a pre-existing language model checkpoint and fine-tune on additional data [11,12,15], although recent work suggests that if domains are sufficiently different, the performance of the fine-tuned model will be limited [16]. The main point of comparison in this thesis will be the performance of pretrained models fine-tuned on in-domain (aviation) data and models trained from scratch on in-domain data. The experiments and results presented in this thesis are preliminary applications of end-to-end deep learning solutions to Automatic Speech Recognition (ASR) and Natural Language Processing (NLP) problems in aviation. The intention is for this work to provide knowledge and possibly checkpoints for the ASR and language models which can be used for downstream tasks such as callsign detection and beam search prediction decoding for ASR models.

2.2 Notation & Terminology

This section is included for the sake of clarity throughout this thesis. In pursuit of that, a few terms and notations are defined below.

Set notation (or set-builder notation)[4] as defined in [17] is used in parts of this thesis, particularly in equations and algorithms, to convey mathematical concepts. The shorthand below is defined

$$A; x := A \cup \{x\}$$

as the union of the set A and a set containing the element x. The equation below can be read as adding the element x to the set A:

$$A \leftarrow A; x$$

The following words (in bold) are used throughout the thesis as they relate to language modeling and are defined concretely as follows:

The definition of words and sentences stem from their definitions in the English language, succinctly, as defined in [18]:

- A **word** is the representation in writing of a sound or combination of sounds that symbolizes and communicates a meaning.
- A **sentence** is a grammatical unit that is syntactically independent that is expressed or understood and contains at least one finite verb.

In terms of the structure of individual samples in a corpus, we will consider each sample in the corpus to be one sentence that is made up of words that are separated by whitespace (one or more spaces) and/or punctuation. Structurally, we consider a word to be made up of one or more characters as defined by the unicode standard [19] and a sentence to be made up of one or more words.

- A **token** shall be defined as the smallest, fundamental component of a sequence; these can be single characters, groups of characters, punctuation, etc. [11, 20, 21]

[4]That is, the notation used to denote sets and associated operations therein as it relates to the field of set theory, in mathematics.

- A **sequence** shall be defined as a set of tokens (distinct from a sentence).

As mentioned above, we consider each **sample** in a corpus to be a sentence, then as the corpus is collated, processed, and tokenized, each sample will be processed into sequences of tokens. These sequences are what will be analyzed and modeled by the language models.

The rules by which words in sentences are broken up into tokens are defined by tokenization algorithms, otherwise known as tokenizers (elaborated further in Section 6.5). While tokens can be individual characters or words as they appear in the original sample, this is not always the case. The intention of most tokenization algorithms is the flexibility to break up unknown words into smaller tokens known to the tokenizer [20, 21].

3 Background & Literature Review

3.1 Transformer Neural Networks

The transformer neural network architecture (shown in Figure 1) was proposed in 2017 for Neural Machine Translation tasks and immediately achieved state-of-the-art performance on language translation tasks [22]. Transformer architectures are extremely effective at learning representations and understandings of languages to predict token probabilities instead of transforming one language into another [11, 12]. Transformers have also been found to be very effective at other NLP tasks such as prompt completion and sentiment analysis among others (i.e. auto-regressive and sequence classification tasks, respectively) [15, 23]. Devlin et al. [11] proposed pretraining methods such as masked language modeling and next sentence prediction to improve natural language understanding for downstream tasks, which was further expanded upon by Liu et al. [12] by training longer, with larger batch sizes, and removing the next sentence prediction task.

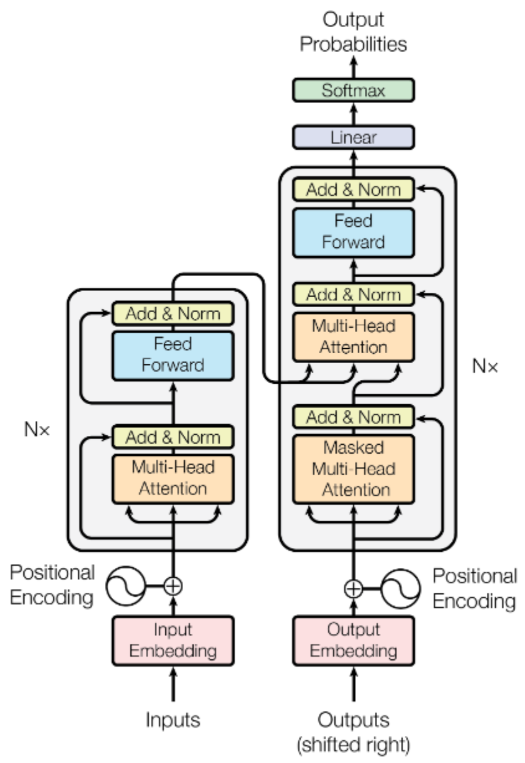

Figure 1: The transformer neural network architecture. Copied from page 3 of Vaswani et al. [22, Fig. 1]

3.2 Tokenizer Algorithms

In language modeling and natural language processing tasks, the out-of-vocabulary (OOV) problem is prevalent and has been addressed by the introduction of tokenizer algorithms. WordPiece and Byte Pair Encoding (BPE) have been used to solve OOV problems while simultaneously minimizing vocabulary sizes and maximizing the likelihood of tokens in sequences [20, 21, 24].

3.3 Language Modeling & Natural Language Processing in Aviation

Various natural language processing methods have been applied to the aviation domain to help deal with miscommunications and try to mitigate safety incidents [25–27]. Some machine learning approaches have been implemented to analyze the text in aviation incident and safety reports to predict contributing factors and topic models [26, 27]. An ASR system with NLP post-processing has also been proposed to analyze Air-Traffic communications and condense significant features (e.g. weather, runway, and heading info) into an XML language structure [25]. Transformer language models such as BERT and RoBERTa have been applied to notice to airmen (NOTAM) messages to perform named entity recognition (NER) tasks, translation tasks (between notations; e.g. NOTAM notation to Airlang to make parsing tasks easier) and reduce the workload for pilots during long-haul flights [28]. Transformer models have also been used for speaker role identification tasks in the aviation domain (e.g. identifying pilot versus controller in communications); specifically, a pretrained BERT model was used and fine-tuned on problem-specific data and compared to other models that performed well at speaker and role identification tasks in general [29].

3.4 Automatic Speech Recognition in Aviation

Automatic speech recognition (ASR) is seeing increased use in aviation for tasks such as transcription, callsign detection, speaker identification, etc. A majority of these approaches use machine learning approaches (as opposed to deep learning) where acoustic, pronunciation, and language modeling is used to transcribe speech [5,29–33]. Language models have been used in the general domain (both neural and machine learning-based approaches) to boost the performance of speech recognition models [4, 34, 35]. Machine learning-based language models, usually N-grams and word lattices, have also been used in aviation alongside semi-supervised approaches to automatic speech recognition to leverage unlabeled data. However, none of the developed methodologies use neural language models for their applications [32, 36, 37].

ASR and speech synthesis technologies have seen use for improving communications and reducing pilot workload as early as 1981 [38–41]. Early approaches focused on developing and improving speech synthesis technologies to shift pilot focus in military and general aviation cockpits to urgent systems or alarms [38,41,42]. ASR system concepts and technologies were also initially developed to reduce pilot workloads and automate simple tasks; these systems were somewhat rigid with fixed vocabularies and keywords, but were able to successfully reduce workload and received positive feedback from pilots [39, 43]. Similar ASR technologies were later applied to Air Traffic Control (ATC) contexts to reduce controller workloads and increase the efficiency of Air Traffic Management (ATM) solutions [25, 44–46]. As ASR technologies have become more sophisticated and more aviation-specific speech corpora have become available [14, 30, 33, 47–51] there have been more applications of ASR to aviation [6, 32, 51–58].

3.5 Differences in Language

The use of the English language in aviation has been shown to be sufficiently different enough to be classified under its own category (aviation English) [9]. The issue with using pretrained checkpoints of transformer-based language models is characterizing the similarity (or, conversely, the difference) between casual and technical English i.e., in this case, the differences between written English, as in books and articles, and the spoken, transcribed English used in aviation. If the two domains are sufficiently different, then training the pretrained models on aviation English data would amount to transfer learning between specific domains. Transfer learning between text domains has been shown to be possible even with data volumes significantly lower than the original data used to train the models [59]. Yadlowsky et al. [16] recently showed that if problem domains are sufficiently different, model pretraining on out-of-domain data will negatively affect model performance.

4 Datasets & Preparation

Four corpora are combined and used for the experiments in this thesis. The datasets are primarily speech corpora intended for automatic speech recognition research in aviation communications, linguistics research in aviation English, or both. All data in all four corpora

ATCC Corpus Properties	
Samples	9,556
Audio (hours)	72.48
Mean Sequence Length	11.90
Std. Sequence Length	7.22
Total Tokens	355,485
Unique Tokens	2,209

Table 1: Data statistics for the Air Traffic Control Complete (ATCC) corpus.

include both recordings and transcriptions, which are audio and text, respectively. The transcriptions are used to pretrain and train the language models and tokenizers (described in sections 6 and 6.5).

The four corpora selected and used for this work are briefly described below. Further analysis is performed in Section 4.7.

4.1 Air Traffic Control Complete

The Air Traffic Control Complete (ATCC) corpus is a speech corpus consisting of audio recordings with corresponding transcriptions. ATCC is a collection of three smaller corpora recorded at Dallas-Fort Worth, Logan International, and Washington National airports and transcribed by current or former air traffic controllers familiar with the respective areas. Audio data was recorded by placing Very High Frequency (VHF) antennae configured to monitor several frequencies at each airport, such as arrival, departure, approach, ground, etc. The types of frequencies observed vary by airport, but the presence of departure, approach, and ground frequencies is relatively consistent across subcorpora [47]. See Table 1 for corpus-specific statistics.

4.2 ATCO2

The ATCO2 dataset is a speech corpus comprising audio communications at Prague and Brno airports in Czechia and corresponding transcriptions. The speech recordings and

ATCO2 Corpus Properties	
Samples	874
Audio (hours)	1.10
Mean Sequence Length	12.28
Std. Sequence Length	5.78
Total Tokens	10,733
Unique Tokens	786

Table 2: Data statistics for the ATCO2 corpus.

transcripts are crowd-sourced from volunteers. The dataset was created and labeled, in part, as an English language detection corpus. However, the labeling also includes transcripts of speech segments, which makes it practical for ASR and language model development in addition to language detection (although in this work, it is only used for ASR and LM development) [48]. See Table 2 for corpus-specific statistics.

4.3 Air Traffic Control Simulation

The Air Traffic Control Simulation (ATCOSIM) corpus is a speech corpus consisting of audio recordings and corresponding transcriptions. The data was recorded at the EUROCONTROL Experimental Centre for air traffic control simulation in Bretigny-sur-Orge, France. In this corpus, only the controllers' voices are included, and therefore, the transcripts only include the controller's side of each interaction. The audio data was transcribed by one person, trained according to the guidelines and formatting requirements of the corpus. After all data was transcribed, it was reviewed and corrected, and any remaining problems were reviewed by an operational air traffic controller and resolved [33]. See Table 3 for corpus-specific statistics.

4.4 ZCU CZ ATC Corpus

The ZCU CZ ATC corpus consists of audio recordings and corresponding transcripts in the Czech airspace. Both the controller and pilot sides of the communications are included

ATCOSIM Corpus Properties	
Samples	9,556
Audio (hours)	10.69
Mean Sequence Length	11.29
Std. Sequence Length	4.17
Total Tokens	107,881
Unique Tokens	827

Table 3: Data statistics for the ATCOSIM corpus.

ZCU CZ ATC Corpus Properties	
Samples	14,435
Audio (hours)	20.58
Mean Sequence Length	10.05
Std. Sequence Length	5.50
Total Tokens	145,107
Unique Tokens	3,100

Table 4: Data statistics for ZCU CZ ATC corpus.

in this data. Experienced transcribers/annotators created annotations, and labeled samples were randomly selected for review during the annotation process. After the dataset was completely labeled, all samples were checked, corrected, and unified [30]. See Table 4 for corpus-specific statistics.

4.5 Data Preparation

All corpora listed above (see Section 4) are created at different times, for various purposes, and by multiple authors. As a result, the data in these corpora all have different formats. ATCC, for example, uses Lisp-style lists as below:

```
((FROM NERA3788) (NUM L02F1-0001)
(TO F1-1)
(TEXT THOUSAND ONE NINETY WE (QUOTE LL) GIVE YOU THAT ON THE SPEED AND
   WE (QUOTE RE) CLEARED FOR THE APPROACH AH NERA THIRTY SEVEN EIGHTY
   EIGHT WE (QUOTE LL) HOLD SHORT OF TWO SEVEN)
(TIMES 1.49 6.57))
```

ATCO2 and ZCU CZ ATC use XML to isolate transcribed segments of speech in the audio recordings, although the format used between the two differs. ATCO2 uses a form like below:

```
<data>
<segment>
<start>3.79</start>
<end>6.85</end>
<speaker>B</speaker>
<speaker_label>OK-PMB</speaker_label>
<text>level one hundred Oscar Kilo Papa Mike Bravo</text>
<tags>
<correct>0</correct>
<correct_transcript>1</correct_transcript>
<correct_tagging>0</correct_tagging>
<non_english>0</non_english>
</tags>
</segment>
</data>
```

ZCU CZ ATC uses a different nesting and labeling style, like below[5]:

```
<Trans audio_filename="e2_ACCU-0agmXf.wav">
<Episode>
<Section type="report" startTime="0" endTime="61.47">
<Turn startTime="0" endTime="61.47">
<Sync time="0.000"/>
[ground]Skyshuttle 1 1 4 0 now descend FL 3 3 0[speaker]
<Sync time="4.790"/>
[air]descending FL 3
<Sync time="6.650"/>
[air]K O Z
<Sync time="9.640"/>
```

[5]Some information is excluded for readability.

```
[ground_|]Austrian 3 2 3 G climb FL 3 4 0 [|_ground][air_|]level 3 4 0
    Austrian 3 2 3 G[|_air]
</Turn>
</Section>
</Episode>
</Trans>
```

Lastly, ATCOSIM uses a text-only format with transcriptions occurring on their own lines in text files:

```
[FRAGMENT] contact geneva one two eight decimal one five good bye
```

Each text file corresponds to the audio file from which the speech was transcribed.

Due to the differing formats above, and the intention to aggregate and use all these transcripts together, all transcripts need to be processed into a common format to be used together.

All data was processed in Python using primarily built-in functions/modules[6]. The text data is extracted from the corpus transcripts by reading each transcript into the main memory of the system according to the format prescribed by the corpus. The text corresponding to the transcript is isolated, copied, and stored to a separate region of memory (a Python list of strings) that corresponds to the original corpus. The resulting isolated transcripts are written to files that correspond to the original corpus as well as combined, shuffled, and written to a file in the standard corpus format (an ASCII text file with one line in the file corresponding to one sample from the corpus). The resulting output is four corpus files that correspond to the original corpora in addition to a fifth corpus file with the samples from all four corpora combined and shuffled for later use[7].

[6]The `re` and `xml` modules were also used, although these are built into most Python distributions by default.

[7]See the `parse_transcripts` functions in the files at `https://github.com/AVanDeBrook/msece-thesis/tree/main/source/data` for implementation details.

4.6 Handling Spelling Errors & Inconsistencies

By the nature of the data labeling process, there is a high likelihood of spelling errors introduced by the annotators during corpora creation. This gets exacerbated by aggregating several corpora with varying controls for correcting spelling errors and inconsistencies. Nearly all of the mistakes are recoverable/correctable, given proper context. For example, "possibility of **tornado's**" should have been "possibility of **tornadoes**", changing the possessive form of "tornado" to the plural form. There are also instances of spelling conventions that are consistent within corpora but become inconsistent when combined.

Manually reviewing every sample in the corpus is unrealistic, considering that there are over 50,000 data samples and would be subject to the same human error that introduced those errors in the first place. Instead, the frequency of token occurrence in the corpora, combined with manual review thereafter, was used to detect and correct the most common errors. Tokens with the lowest frequency across corpora (and lowest number of occurrences) were manually reviewed and corrected where necessary. Tokens that needed to be corrected typically occurred five times or less per corpus although all tokens with occurrences ranging from 1-50 were reviewed. The relative frequency of those tokens varies by corpus (due to the differing amont of tokens in total for each corpus) The frequency of occurrence of a token is calculated using equation (1), below

$$f_w = \frac{\phi_w(w)}{D_{tokens}} \tag{1}$$

where f_w is the frequency at which a token occurs in a corpus, $\phi_w(w)$ is the number of times a token, w, occurs, and D_{tokens} is the total number of tokens in the corpus.

An $A \rightarrow B$ mapping system was used to correct the errors, where A is the erroneous token and B is the corrected or replaced token. As the text is parsed from the corpora, each sample is analyzed. If an erroneous token, A, is found, it is replaced with the corrected version, B. 22 unique errors were detected and subsequently corrected or removed using

Phenomenon	Definition
Hesitations	Instances of spoken words for which the specific realization is not defined (usually some special token is used instead).
Partial & cut-off words	Partially pronounced words are indicated by a hyphen character where the pronunciation of the word is missing.
Filled pauses	Hesitations with a verbal component that has been described or otherwise defined by the corpus/annotator.

Table 5: Speech/transcription phenomena with corresponding convention for transcribing or translating, depending on the corpus being processed.

Phenomenon	Convention for representation
Hesitations	[HES]
Partial & cut-off words	-
Filled pauses (open-mouth)	uh, um, ah
Filled pauses (closed-mouth)	hm, mhm

Table 6: Conventions for transcript representation/translation of phenomena to keep transcripts consistent across corpora.

this method. However, multiple instances of those errors typically occurred.

The mapping system described above is also used to make corpora conventions to be consistent across corpora. Three areas are addressed here: hesitations, cut-off, or partial words, and filled pauses within speech. These areas are defined differently between the four corpora, which makes unifying the transcripts difficult. Hesitations, for example, are defined by one corpus as incomplete or non-English speech sounds and left completely undefined by another (despite annotators making notes about hesitations). The way in which hesitations, pauses, and partial words are transcribed differs between corpora as well. Two corpora prescribe special tokens to these phenomena, such as “[hes]”, “[hesitation]”, “<pause>”, or similar. In contrast, the others transcribe the data as it sounds (i.e., “uh”, “eh”, or “er” for filled pauses and “cir-” or “cir+” for instances that were cut-off mid pronunciation). To facilitate the unification of the transcripts for all four corpora, the conventions for transcribing these phenomena are redefined in Table 5.

A special token represents hesitation when no other context is present regarding the realization of hesitation. If other corpora use a special token to express hesitations, it is translated to match the convention in Table 6. Partial/cut-off words are represented with a hyphen (-) at the point where the pronounced word is cut off. For example, if the last syllable of "approaching" is cut mid-transmission, it would be represented in the transcript as "approa-" or, likewise, if the first syllable is cut off, it would be written as "-roaching". Lastly, if there is enough information in the original transcript to indicate the realization of a hesitation, it is represented by the most appropriate token from "Filled pauses" in Table 6.

One of the intended downstream use cases of the language models developed from this work is decoding speech recognition predictions into transcripts, so including as much data as possible regarding spoken word realization is useful given the context. The `[HES]` token is used as a fallback token if no knowledge about the utterance can be extracted from the transcript. This usually occurs when the original transcript did not include any detail regarding the special case of an utterance, for example, an `[UNINTELLIGIBLE]` token to mark unintelligible speech, or `[UNK]` to mark an unknown word or utterance that could not be transcribed. The `[HES]` token is therefore used as a way of unifying the different approaches to mark hesitations, unintelligible speech, etc. when a special token was used with no indication of the realized speech.

4.7 Data Analysis

To better understand the above four corpora, we calculate and tabulate their relevant corpora statistics in Table 7. This includes the number of samples in the corpus, the mean sequence length per sample, the number of unique tokens, and the total number of tokens in the corpus. The same statistics were also calculated across corpora since all four corpora will be combined for training and testing the language and speech recognition models. Note that this is based on the raw data after loading and unifying the corpora before preprocessing for model training/testing and outlier removal.

Corpus	**Samples**	**Mean Sequence Length** ($\pm\sigma$)	**Unique Tokens**	**Tokens**
ATCC	29,862	11.90 ± 7.22	2,209	355,473
ATCO2	874	12.28 ± 4.12	786	10,733
ATCOSIM	9,544	11.23 ± 4.12	823	107,153
ZCU CZ ATC	14,435	10.05 ± 5.50	2,983	145,107
Total	54,715	11.30 ± 6.36	5,040	618,466

Table 7: The number of samples, mean sequence length, number of unique tokens, and total number of tokens by corpus, including the cumulative total across all corpora.

The ATCC and ZCU CZ ATC corpora make up a significant portion of the data, ATCC alone makes up more than half (approximately 54%) of the combined data, and ATCC and ZCU CZ ATC together make up approximately 81% of the data. The sequence lengths of the samples are fairly consistent across corpora with at most about an 18% percent difference between sequence lengths (calculated using equation (2), below). The number of unique tokens does not scale additively to the combined corpora, which is expected since there will be an overlap of tokens. Aviation English generally conforms to and follows International Civil Aviation Organization (ICAO) or Federal Aviation Administration (FAA) phraseology standards[8], so we expect many terms to be shared across corpora except geographic and location-specific terms, such as city, landmark, airport, and airline names, which vary across regions.

$$\text{PD} = \frac{|V_1 - V_2|}{\frac{V_1+V_2}{2}} \times 100 \tag{2}$$

In the equation above, V_1 and V_2 are the two numbers being compared.

4.8 Outlier Detection and Analysis

The Local Outlier Factor (LOF) algorithm [60] was used to detect and analyze outliers in the corpora. The sequence lengths of the data samples were used to determine outliers.

[8]The applicable standard depends on the airspace and country. The ICAO and FAA standards are similar but different, so both are included.

Outlier Properties	
Mean Sequence Length	47.30
Std of Sequence Length	11.82
Minimum Sequence Length	38
Maximum Sequence Length	73
# Samples	47

Table 8: Mean, standard deviation, minimum, and maximum sequence length of samples detected as outliers.

Since all four corpora are aggregated, LOF analysis was performed on the aggregated data instead of on the individual corpora.

Sci-Kit Learn's [61] implementation of the LOF algorithm was used with a value of $K = 35$ for the K-Nearest Neighbors algorithm; the value for K is arbitrarily chosen, partially through trial and error and the size of the clusters for sequence lengths. Minkowski distance is used for the distance metrics and is defined as

$$d(x, y) = (|x - y|^p)^{\frac{1}{p}} \tag{3}$$

where x and y are the points between which the distance is being measured and p is the order on which to calculate the distance[9]. $p = 2$ is chosen in this case to make the Minkowski distance equivalent to Euclidean distance:

$$d(x, y) = \sqrt{|x - y|^2} \tag{4}$$

A total of 47 samples were detected as outliers and removed for analysis. They are summarized in Table 8.

Although all of the outliers have longer sequence lengths than the typical sequence lengths of the other samples in the data, going through the outlying samples manually and reviewing each reveals that, for the most part, the samples are typical albeit somewhat complex communications. Except for one outlier, all of the samples labeled outlying by the

[9]For the LOF algorithm, p can be arbitrarily chosen.

LOF algorithm were kept in the data. Examples of the outliers are listed in Appendix A in Section 10, including the removed sample.

5 End-to-End ASR Models & Experiment Results

Automatic speech recognition (ASR) or simply speech recognition is the process of translating spoken language to text i.e. transcribing spoken language. There have been a myriad of approaches to ASR problems, including, but not limited to rule-based, machine learning, and deep learning methods. For the purposes of this work, we will focus primarily on end-to-end deep learning approaches to ASR tasks with a specific focus on transcribing spoken language in aviation (following from the data accumulated, processed, and analyzed in sections 4, 4.5, and 4.7).

5.1 Input Representation

Although the internal processing of data varies between architectures, the representation of the models' input for the models described below. ASR models have seen much success and performance gains for speech-related tasks such as speech and speaker recognition through the use of Mel-frequency Cepstral Coefficients (MFCCs) [62]. Since ASR is not the main focus of this thesis, the process for computing the MFCCs will only be described from a high level as it is described in [62]. Readers should refer to [62] for a more detailed, matrix-based approach for computing MFCCs. Given an audio signal:

1. Multiply the audio signal by a tapered window such as a Hamming or Hanning window to obtain windowed speech frames
2. Pad the windowed audio signal with zeroes to facilitate the fast Fourier transform of the signal

3. Multiply the zero-padded windowed speech frames by the twiddle factor matrix to obtain the discrete Fourier transform (DFT) coefficients

4. Multiply each DFT coefficient with its conjugate to obtain the power spectrum of the signal

5. Pass the signal through a triangular filter bank with linearly spaced filters in the Mel-scale. Obtain the log-energy output by taking the log of the output — the filter bank log energy

6. Finally, multiply the filter bank log energy from the previous step by the discrete cosine transform matrix to obtain the MFCCs

The MFCCs are then used as input to the ASR models. It should be noted that MFCCs are not the only input representation available for ASR models, just the most popular choice among the models used here. Each model outputs a probability distribution of characters or tokens (depending on the vocabulary the ASR model is initialized with) over a segment of audio.

5.2 Training Objective

The ASR models listed below both use a Connectionist Temporal Classification (CTC) training objective [63]. For a dataset, D, made up of samples and labels, x and y, respectively, defined as

$$D = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\} \tag{5}$$

where N is the number of samples in the dataset. The goal of the ASR model training procedure is to predict the target, y, based on the input sequence, x. This can also be written as maximizing the likelihood of $p(y|x)$. More formally, the CTC objective function can be written as

$$O^{ML}(D, \mathcal{N}) = -\sum_{(x,y)\in D} \ln p(y|x) \tag{6}$$

where $\mathcal{N}$ is the network being optimized to maximize the likelihood of predicting $p(y|x)$. Put simply, the goal of the ASR network is to maximize the log-likelihood of predicting a label, y, given an input, x.

The novelty of the CTC objective is in the fact that the MFCC frames do not need to be aligned with the letter-based tokens to train a network. Instead, the output layer of the network has $L + 1$ units, where L is the alphabet/vocabulary of the network. At each time step in the audio signal, the network will predict either a token from its vocabulary or a blank token, hence the extra unit in the output layer. Repeated predictions and blank tokens are removed from the prediction, then scored against the ground truth label for the audio signal [63]. Note that Graves et al. [63] defined the tokens predicted at each time step as letters and the vocabulary, correspondingly, as the letters *A* through *Z* of the alphabet.

5.3 Jasper

Jasper is a family of deep convolutional neural networks (CNNs) developed by NVIDIA as end-to-end ASR models to replace traditional ASR models that use separately learned components for each stage of the pipeline (acoustic, pronunciation, language modeling, etc.) [3].

Jasper models, shown in Figure 2 are designed with B blocks and R sub-blocks and are named accordingly as *Jasper BxR* models. The Jasper architecture also contains four extra convolutional blocks, one for preprocessing, shown at the bottom of Figure 2, and three for postprocessing, shown at the top of Figure 2. Each convolutional sub-block contains one dimensional convolutions. Li et al. [3] trained and tested a Jasper 10x5 model that produced state-of-the-art results on the LibriSpeech, Wall Street Journal, and Fisher+Switchboard speech corpora.

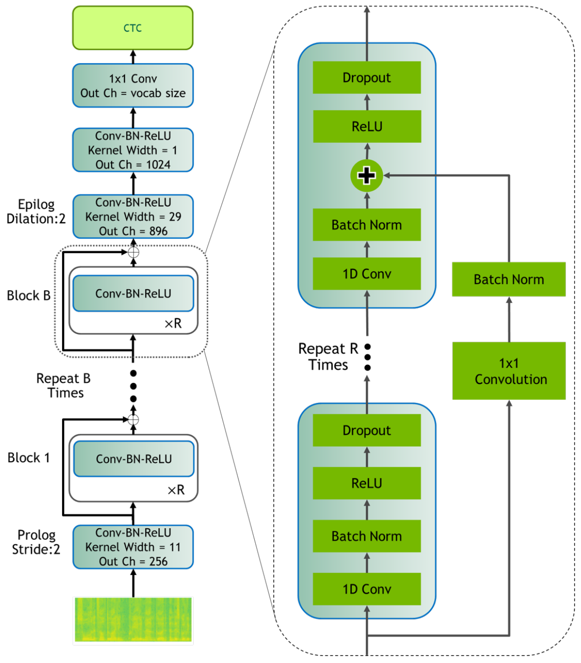

Figure 2: Block diagram of the Jasper architecture. Copied from Li et al. [3].

5.4 QuartzNet

The QuartzNet architecture is based on Jasper with some key modifications, namely, that the one dimensional convolutions are replaced with "one dimensional time-channel separable convolutions", as shown in Figure 3. As with Jasper, QuartzNet models are denoted as *QuartzNet* BxR with B and R representing the number of blocks and sub-blocks, respectively. However, blocks are repeated S times and have a certain number of input and output channels, c_{in} and c_{out}, respectively. Time-channel separable convolutions are separated into two components: (1) convolutional layers that operate on each audio channel independently across time frames and (2) convolutional layers that operate on time frames independently across all audio channels [4].

In Kriman et al. [4] experiments with QuartzNet were conducted on the LibriSpeech and Wall Street Journal speech corpora and achieved near state-of-the-art performance on both. A transfer learning experiment was also performed (transferring between corpora). The model was initially trained on LibriSpeech and Mozilla Common Voice, then fine-tuned on the Wall Street Journal corpus and also achieved near state-of-the-art results [4]. These

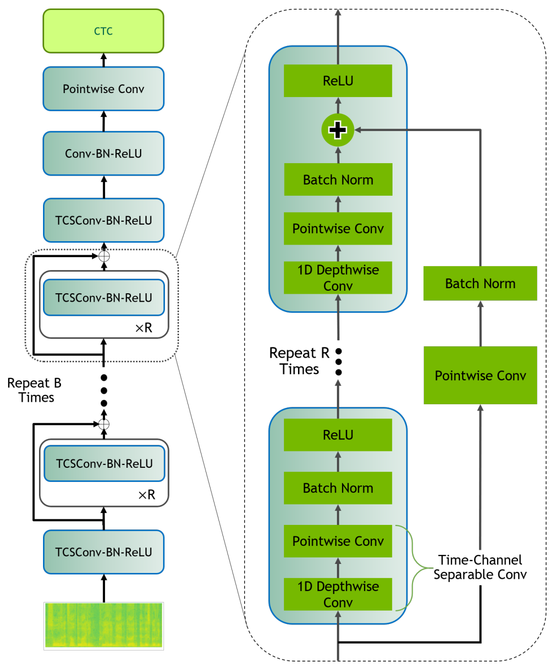

Figure 3: Block diagram of the QuartzNet architecture. Copied from Kriman et al. [4].

results suggest that the QuartzNet architecture not only performs well on in-domain tasks, but can generalize and fine-tune well on related data.

5.5 Experiment Results

This section will summarize the experiment setup, evaluation metrics, and final results of the two best performing ASR models. Two Jasper models with 10 blocks and 5 sub-blocks and two QuartzNet models with 15 blocks and 5 sub-blocks were trained from scratch and fine-tuned from pretrained checkpoints; four models were evaluated in total. Each model used a vocabulary of 28 tokens; the English alphabet, an apostrophe, and a blank label.

5.5.1 Performance Metrics

This section defines the metrics used to track the performance of the ASR models during training and the performance of the models on the test partitions of their datasets.

Word Error Rate (WER). This is, simply, the number of words predicted incorrectly by the model. WER is based on the edit distance between the models prediction and the

Parameter	Value
β_1	0.8
β_2	0.5
ϵ	1×10^{-8}
Learning rate	0.01
Weight Decay	0.001

Table 9: NovoGrad optimizer parameters for the Jasper and QuartzNet ASR model training procedures.

prediction target [64]. Let N, S, I, and D be the number of words, substitutions, insertions, and deletions, respectively. Then the WER is calculated as

$$\text{WER} = \frac{S + D + I}{N} \times 100 \tag{7}$$

In this case, we calculate WER as a percentage by multiplying by 100. Note that it is possible for the WER to be greater than 100% if $S + D + I > N$. This can occur, for example, if the model predicts more words than are actually present in the audio signal.

5.5.2 Optimizer

The NovoGrad optimizer [65] is used with the parameters shown in Table 9.

The Jasper model trained from scratch performed best overall with about a 15% WER on the test data. The pretrained QuartzNet model that was fine-tuned on the aggregated data performed second best among the evaluated models reaching a WER of about 30%. The best-performing ASR models are summarized in Table 10. These tests were run using a greedy decoding method, meaning that at each time step, the most active neuron i.e., that with the highest value after softmax was chosen as the prediction. These results can likely be improved further by adjusting the parameters of the model. However we believe the most significant boost in performance can be achieved using a well-performing language model with a beam search decoding algorithm. Beam search-based decoding methods use a language model and a beam search algorithm to determine predictions at each time step.

Model	WER (%)
Jasper (scratch)	15.3
QuartzNet (pretrained)	30.2

Table 10: WERs of the best-performing ASR models on the test partition of the corpus.

The most commonly used language model for these types of decoding methods is an N-gram and is what PyTorch's `CTCDecoder`[10] [66] and NeMo's `BeamSearchDecoderWithLM` [67] use to decode predictions. The purpose of language models in beam search decoding algorithms is to estimate the likelihood of the decoded sequence occurring based on the language models' knowledge of sequence structures. The beam search algorithm is then used to select tokens to maximize the likelihood of the sequence occurring. N-grams are left-associative language models (see Section 6.1; assuming the model is being applied to the English language in this case), which is an inherently limiting factor. A neural language model capable of representing sequences bidirectionally, such as those covered in this work, is a significantly more robust method of language modeling in speech recognition and has been used in the past to improve word error rates [4, 35]. At least one instance of a deep learning-based language model being used for beam search decoding in aviation applications. However, it did not perform well, likely due to the small size of the dataset that it was trained on [13]. In the next section, the language model architectures, experiments, and results are defined for the purpose of improving ASR model performance through the use of language models and downstream tasks such as callsign detection.

6 Language Models and Experiment Results

This section introduces the chosen language model architectures and their pretrained checkpoints. Two transformer-based architectures, BERT and RoBERTa, and one statistical/machine learning-based approach: N-gram language modeling.

[10]More specifically, the `torchaudio` package of PyTorch.

The transformers are pre-trained using the same masked language modeling (MLM) objective presented in Devlin et al. [11] to train BERT for bidirectional encoding representation and likewise in Liu et al. [12] for training RoBERTa. Given the results of the RoBERTa pretraining and downstream task training thereafter, it was decided to eliminate the next sentence prediction (NSP) task that was used to pretrain BERT. MLM and NSP pretraining are explained more in-depth in Section 6.3 below.

6.1 N-Gram

An N-gram language model estimates the probability of a token occurring in a sequence based on the N tokens preceding or succeeding that token. In other words, for a sequence, X, of T tokens

$$X = w_1, w_2, w_3, ..., w_T \tag{8}$$

the probability of tokens occurring in order, in a sequence can be represented using the chain-rule of probability:

$$\begin{gathered} P(X) = P(w_1)P(w_2|w_1)P(w_3|w_2, w_1)...P(w_T|w_{T-1}, w_{T-2}, ..., w_1) \\ P(X) = P(w_i)\prod_{i=2}^{T} P(w_i|w_{i-1}, w_{i-2}, ..., w_1) \end{gathered} \tag{9}$$

Except for the first token in the sequence, the probability of a token occurring at a specific point depends on the tokens preceding it[11]. Due to this fact, N-grams are sometimes called left-associative models (in a left-to-right language).

The probability of a token occurring at all is determined statistically as the number of occurrences in a corpus divided by the total number of tokens.

$$P(w) = \frac{\phi_w(w)}{D_{tokens}} \tag{10}$$

[11]The most common use case for an N-gram model is the preceding tokens.

where $P(w)$ is the probability of a token, w, occurring anywhere in the corpus, $\phi_w(w)$ is the number of times a specific token appears in the corpus, and D_{tokens} is the total number of tokens in the corpus, D. Computationally, determining the probabilities of the individual tokens in the corpus is typically referred to as training or fitting the N-gram model.

N is a predetermined value based on model or application requirements. While it can technically take any value from one to infinity, its usability becomes very limited at the minimum sequence length in the corpus and unusable at any value at or above the maximum sequence length. Depending on implementation, the time and resources required to train a model with a large value of N can be very expensive. Common values for N are one, two, and three, sometimes called unigrams, bigrams, and trigrams, respectively.

6.2 Transformer

The transformer neural network uses an Encoder-Decoder architecture with multi-head self-attention layers throughout the network. Details of the general design of the network will be laid out here in more detail, including the construction of the layers, sub-layers, and operation of the attention mechanism. The design discussed here is identical to the network described in *Attention Is All You Need* [22] and is the base of the design of BERT and RoBERTa models discussed in Sections 6.3 and 6.4, respectively.

The Encoder Stack. The encoding layers (referred to as the "Encoder Stack"; see Figure 4) are composed of N identical layers, each containing two sub-layers: (1) a multi-head self-attention mechanism and (2) a position-wise fully connected feed-forward network. Each sub-layer includes a residual connection that is added to the output of the sub-layer and normalized. All sub-layers produce outputs of dimension d_{model}. Vaswani et al. [22] used $N = 6$ encoding layers and $d_{model} = 512$ output dimensions. The feed-forward network uses an inner dimension of $d_{ff} = 2048$

The Decoder Stack. The decoding layers are also comprised of N layers (the "Decoding Stack"; see Figure 5) with three sub-layers for each layer: (1) a masked multi-head attention

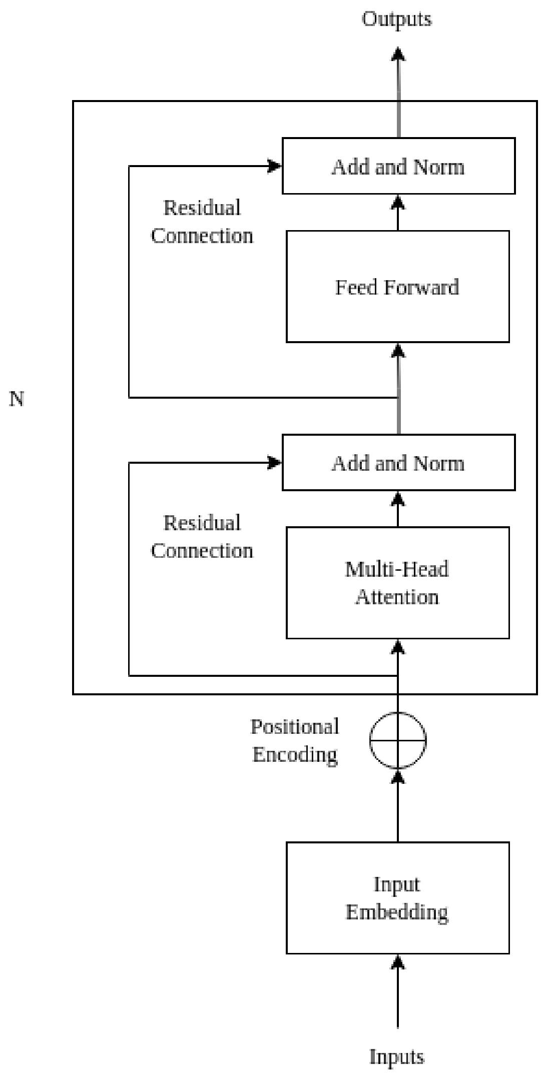

Figure 4: The Encoder Stack of the transformer architecture. Modified from [22].

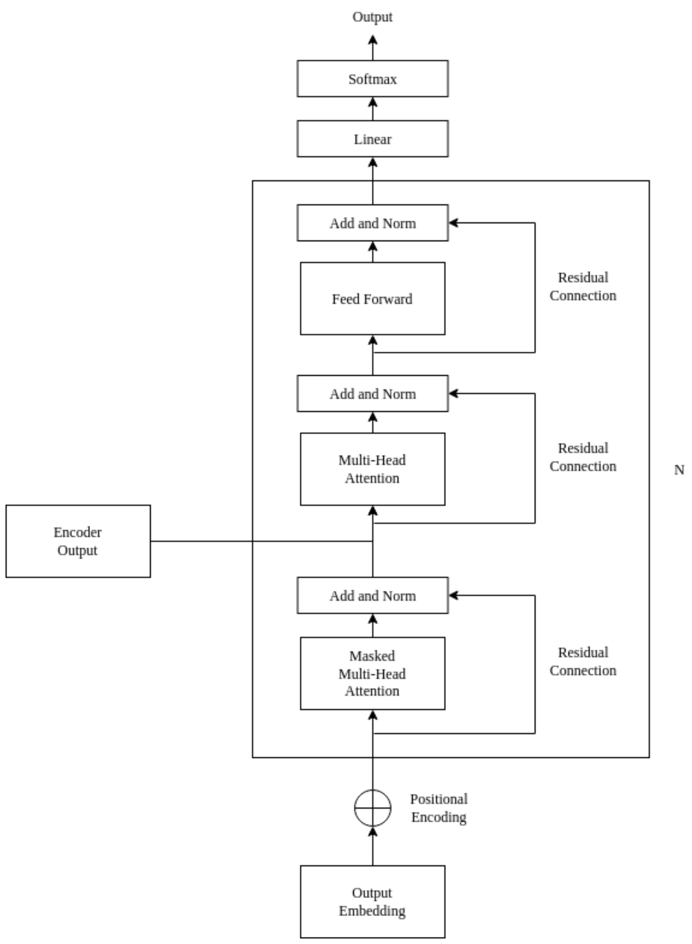

Figure 5: The Decoder Stack of the transformer architecture. Modified from [22].

sub-layer, (2) a multi-head attention sub-layer over the outputs of the Encoder Stack, and (3) a position-wise fully connected feed-forward network. The last two sub-layers are identical to those in the Encoder Stack. Each sub-layer also includes a residual connection that is added to the output of the layer and normalized. The first multi-head attention layer is modified to a masked multi-head attention layer, masking off tokens greater than the ith position. This, in combination with the output embeddings being shifted right by one position, ensure that predictions for a token position, i, are only dependent on outputs for positions less than i [22].

The Multi-Head Attention Mechanism. By the simplest of terms, attention mechanisms can be described as mapping a query and a set of keys and values to an output. In the case of the transformer described by Vaswani et al. [22], a version of attention called "Scaled Dot-Product Attention" is used. The query and keys are represented as vectors Q and K, respectively, with dimensions of d_k. The values are represented by a vector, V, with a dimension of d_v. A dot-product of the query and keys is computed, scaled by $\sqrt{d_k}$. A softmax function is then applied to the output, given by equation (11).

$$\text{softmax}(x) = \frac{\exp(x_i)}{\sum_{j=1}^{N} \exp(x_j)} \quad \forall i = 1..N \tag{11}$$

Where x is a vector of dimension N. Scaled-dot product attention can then be given by equation (12), below.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{12}$$

Multi-Head attention is defined as multiple scaled-dot product attention layers, running in parallel. More specifically, the queries, keys, and values are linearly projected h times with different learned linear projections to d_k, d_k, and d_v dimensions, respectively. The attention function is applied to each projection resulting in output vectors with dimensions of d_v. These are concatenated and linearly projected once more to produce the output for the multi-head attention function [22].

$$\begin{gathered}\text{MultiHeadAttention}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h) W^O \\ \text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{gathered} \tag{13}$$

Where W_i^Q and W_i^K are weight matrices for the projections of dimensions $d_{model} \times d_k$, W_i^V is a weight matrix for the projection of dimension $d_{model} \times d_v$ and W_i^O is a weight matrix for the projections of dimension $hd_v \times d_{model}$.

Vaswani et al. [22] used $h = 8$ attention heads and $d_k = d_v = d_{model}/h = 64$.

Token Embeddings. Input and output tokens are converted to learned representations of dimension d_{model} by the model [22].

Model	Layers (N)	Hidden Layer Size (d_{ff})	Attention Heads (h)
AIAYN	6	2048	8
BERT$_{base}$	12	768	12
BERT$_{large}$	24	1024	16

Table 11: Number of layers, hidden layer sizes, and number of attention heads for the base and large versions of the BERT model compared to the model used in *Attention Is All You Need* (AIAYN) [11, 22].

6.3 BERT

The purpose of BERT (Bidirectional Encoder Representations from Transformers) is to represent tokens in text sequences based on the tokens preceding and following the token in question. Whereas an N-gram model represents token probabilities based on the N tokens preceding a specific token or, conversely, the N tokens succeeding said token, BERT models can predict the probability of a token based on both preceding and succeeding tokens; the bidirectionality novelty.

BERT is based on the transformer neural network described in Section 6.2 and uses an implementation that is nearly identical to that of the original model [11, 22]. The notable differences between the original implementation and the implementation of the base and large BERT models are shown in Table 11.

Representation of Inputs. BERT uses a summation of the token embeddings (token IDs), segment embeddings (zero or one to represent the first or second sentence in a sentence pair), and position embeddings (the index of the token in the sequence) to form model inputs. Additionally, the WordPiece tokenizer algorithm (detailed in Section 6.5.2) is used to segment tokens. The maximum sequence length of the model is 512 tokens and the size of the learned embeddings corresponds to d_{model} (768 for BERT$_{base}$ and 1024 for BERT$_{large}$) [11].

Masked Language Modeling. The ability of BERT to represent text in a bidirectional context is achieved through the Masked Language Modeling (MLM) pretraining task [11]. MLM pretraining chooses a certain percentage of token positions at random as labels for prediction; those tokens are then replaced with either a “mask” token, a random token, or the original token according to preconfigured probabilities (BERT used 80% for the mask token, 10% for random tokens, and 10% for the original/unchanged token). During training, the masked sequences are used as input to the model, and the original is used as the prediction target. The model prediction is scored against the target using cross-entropy loss.

Next Sentence Prediction. While MLM conditions the model to represent tokens within sequences and predict the most probable token based on its position in the sequence, NSP is meant to condition the model to understand relationships between sentences/sequences to support downstream tasks. Since the BERT training set consists primarily of literary works and excerpts from Wikipedia, the model simply predicts whether two sentences appear together (one after the other) in the original text, making this a binary classification task as opposed to MLM, which is a multi-class classification task. Instead of a single sequence, the input consists of two sequences separated by a “separation” token and beginning with a “classification” token; these are represented by `[SEP]` and `[CLS]`, respectively in Devlin et al. [11].

The special tokens specifying the beginning and end of input sequences are shared between tasks for symmetry between pretraining tasks. For both MLM and NSP objectives, input sequences always start with a classification token and end with a separation token (see Figure 6). Likewise, NSP inputs always begin with a classification token, and each input sequence is concatenated with a separation token terminating each sequence (see Figure 7 and equation (14)).

$$\begin{gathered}\text{[CLS], the, quick, brown, fox, [SEP]}\\ \text{[CLS], the, quick, brown, fox, [SEP], jumps, over, the, [SEP]}\end{gathered} \tag{14}$$

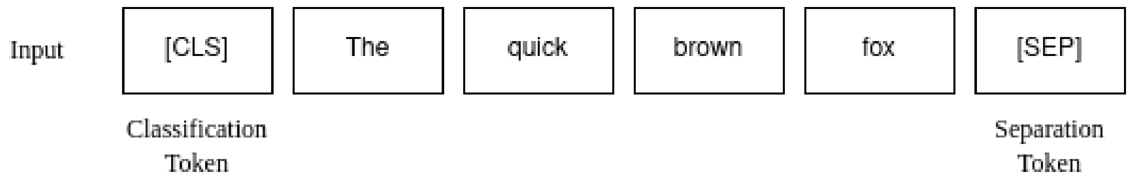

Figure 6: An example of an input sequence for a masked language modeling pretraining task. Modified from Devlin et al. [11].

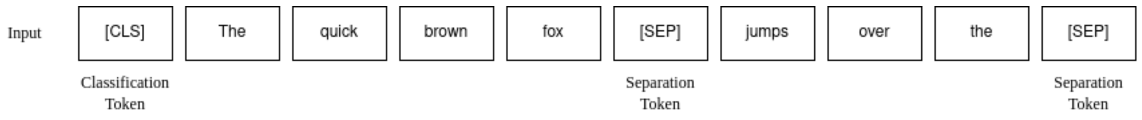

Figure 7: An example of an input sequence for a next sentence prediction pretraining task.

6.4 RoBERTa

As the name implies, the Robustly Optimized BERT Pretraining Approach (RoBERTa) is an iteration on BERT (described in Section 6.3, above). The results from Liu et al. [12] indicate that BERT was initially significantly undertrained, so by training longer, with larger batch sizes, on longer sequences, and dynamically changing the masking pattern applied to the MLM pretraining task, the performance of BERT was improved. The new checkpoint of this version of BERT was, therefore, called RoBERTa. The architecture and parameters of the RoBERTa models are identical to that of BERT.

The pretraining methodology introduced by RoBERTa eliminated the NSP pretraining task originally used in BERT and opted to increase the quantity of training data along with the batch sizes and train the model longer. Lastly, there is a small difference in the construction of input sequences as compared to BERT; input sequences always begin with a classification token (`[CLS]`), and individual segments are separated by a separation token (`[SEP]`). Rather than having sequences end with a separation token, they are terminated with an end-of-sequence token (`[EOS]`). See equation (15) modified from Liu et al. [12], below.

$$\text{[CLS]}, x_1, x_2, ..., x_N, \text{[SEP]}, y_1, y_2, ..., y_M, \text{[EOS]} \tag{15}$$

6.5 Tokenization Algorithms

Tokenization is the process of separating or segmenting words in sentences into their component "tokens". The tokenized version of a sentence is referred to as a sequence. The rules by which the tokens are created or segmented depend entirely on the algorithm used. The simplest and most intuitive way to do this is by creating tokens based on whitespace, i.e., each word in the corpus is treated as a token[12]. While this is simple and works in theory, it does not consider that not every word in a language will occur in a corpus. The inevitable possibility of a tokenizer seeing a word not in the original training corpus is known as the out-of-vocabulary (OOV) problem [20]. Several different algorithms have addressed the OOV problem. For brevity's sake, only the tokenization algorithms used by the models in this thesis are defined and explained.

6.5.1 WordLevel

The WordLevel [68] tokenization algorithm, mentioned briefly as an introductory example above, is the most technically simple algorithm among those introduced in this section. Tokens are extrapolated from words in a sentence based on the whitespace separating each word. This means that contractions, hyphenations, etc. are treated as their own words, i.e. "you", "you're", and "you've" are all seen as unique, distinct words tokenized and mapped to their integral representations.

An index of the words that appear in a corpus must be created to map individual tokens to numerical values or IDs. The numerical values for each token are derived from this index. For the WordLevel algorithm, this process is referred to as the tokenizer model training (see Algorithm 1).

The first step of the training procedure is to index any specified special tokens, such as mask, classification, or separation tokens (this is why special tokens often have low-valued token IDs). The second step is to split the sentences in the training data into individual

[12]The Huggingface Tokenizer library implements this as the `WordLevel` tokenizer.

Algorithm 1 Training procedure for the WordLevel tokenizer.

```
$D = \{s_1, s_2, ..., s_N\}$        ▷ $D$, $s_i$, and $N$ are corpus, sentence, and corpus length
$V \leftarrow \{\varnothing\}$        ▷ $V$ is the tokenizer vocabulary and is initialized to the empty set

for $s \in D$ do
    Split $s$ into words using whitespace as a delimiter such that
    $W = \{w_1, w_2, ...w_M\}$        ▷ $w$ and $M$ are words and sentence length

    for $w \in W$ do
        if $w \notin V$ then
            $V \leftarrow V; w$        ▷ Add $w$ to the vocabulary
        end if
    end for
end for
```

Tokenization stage	Representation
Input string:	“the quick brown fox jumps over the lazy dog”
Tokenized string:	[“the”, “quick”, “brown”, “fox”, “jumps”, “over”, “the”, “lazy”, “dog”]
Integral mapping:	[5, 16, 9, 12, 13, 15, 5, 14, 11]

Table 12: A simple example of the tokenization process of the WordLevel tokenization algorithm.

words. This is done by splitting each string based on whitespace; in other words, the tokenizer treats sentences as lists of words delimited by space characters. Lastly, the tokenizer iterates through each word in each sentence, indexing new words as it finds them until all sentences in the training corpus have been analyzed. Token IDs are then derived based on the word’s position in the tokenizer index or vocabulary.

Table 12 shows the different representations of the input sentence over the course of the tokenization process.

The algorithm’s simplicity speaks for itself and serves as a good introduction to tokenization and an illustrative example of the OOV problem. Note that the values of the integers representing the tokens are determined based on the order in which the tokens are seen during the training process (special tokens are defined and, therefore, seen first), so these results are completely reproducible as long as the data appears in the same order.

The tokenization of words with shared roots but different suffixes is shown in Table 13.

Tokenization stage	Representation
Input string:	“you you’re you’ve”
Tokenized string:	[“you”, “you’re”, “you’ve”]
Integral mapping:	[18, 19, 20]

Table 13: The results of tokenizing three words with the same roots and differing suffixes.

Tokenization stage	Representation
Input string:	“you’ll”
Tokenized string:	[“[UNK]”]
Integral mapping:	[0]

Table 14: Results of running the WordLevel tokenizer on an unknown word similar to some of those in the training data.

The tokenizer sees all three words as independent and unique words since they are mapped to their own integer representations.

The tokenizer used to generate the examples in tables 12 and 13 was trained on two sentences, notably lacking the word “you’ll”. Running the tokenizer on the word “you’ll” alone results in the output in Table 14. Although the word shares a root with some of the words in the training data (“you”, “you’re”, and “you’ve”) and is syntactically very similar, the tokenizer does not recognize the word at all (represented by the “[UNK]” or unknown token). This word would be considered out-of-vocabulary since it did not appear in the training data and thus reveals one of the major downsides of this algorithm, as mentioned in section 6.5. This problem can be alleviated to some extent by modifying the preprocessing strategy of the tokenizer to partition contracted words into their component parts, e.g. the word “you’re” would be partitioned to `[‘‘you’’, ‘‘’’’, ‘‘re’’]`, however, it eventually reappears, for example, when using the past, present, and future tenses of a verb (e.g. “go” and “goes” in the training data, then encountering the future tense: “going”).

6.5.2 WordPiece

Schuster and Nakajima [24] developed the WordPiece algorithm as a word segmentation algorithm for language modeling in Japanese and Korean voice search systems. It is further used by Wu et al. [20] for neural machine translation tasks. The algorithm demonstrated proficiency and increased performance for the models used in both tasks and effectively addresses the OOV problem.

One of the primary stipulations for this algorithm is effective and efficient handling of OOV words such that none are produced during normal operation. To achieve this, during training, the vocabulary of the tokenizer is initialized to a basic set of characters; since WordPiece is designed to be language agnostic, the initial vocabulary is specified as all basic unicode characters in addition to all ASCII characters.

The training procedure works by iterating over the vocabulary, combining two word units to maximize the likelihood over the training data, and repeating until one of two stop conditions are reached:

1. The predefined word limit is reached
2. The increase in likelihood falls below the predefined threshold

The training procedure is simple but computationally expensive. Schuster and Nakajima calculated the algorithm's time complexity at $O(K^2)$ where K is the current size of the vocabulary [24]. Due to the high time complexity of the algorithm, the authors suggest the following considerations to reduce the computational complexity:

- Test only pairs that actually exist in the training data
- Test only pairs with a significant chance of being best (high prior likelihood)
- Combine several clustering steps into a single iteration (for groups of pairs which don't affect each other)

Tokenization stage	Representation
Input string:	“the quick brown fox jumps over the lazy dog”
Tokenized string:	[“the”, “quick”, “brown”, “fox”, “jumps”, “over”, “the”, “lazy”, “dog”]
Integral mapping:	[59, 98, 95, 88, 97, 91, 59, 90, 87]

Table 15: A simple example of the tokenization process of the WordPiece algorithm.

- Only modify language model counts for the affected entries

Table 15 shows an example of a WordPiece tokenizer model output on a sentence that was seen during the training procedure (trained on the same data as the WordLevel tokenizer in Section 6.5.1). Since this sentence was in the training data, the segmentation is identical to that of the WordLevel tokenizer in Table 12, however, tokenization of similar words (e.g. words with shared roots or similar contractions) are handled differently, as in Table 16. Since each word has a shared root word in the contractions (as well as an apostrophe separating the contraction), the tokenizer segments each word and represents them as groups of small, more common tokens (thus with higher probabilities of occurring). Additionally, the WordPiece algorithm effectively handles OOV words. For example, the OOV tokens in Table 17, with a shared root and similar contractions as those in Table 16, which appear in the training data. The base of the contraction, “you”, occurs in the training data and the apostrophe in the contraction, so the tokenizer segments those parts of the word as tokens. The last part of the word, “ll”, does not appear at all in the training data, so it is broken down into its component characters, which are represented as “l” (beginning of a word) and “##l” (segmented part of a word that should be concatenated to the previous token when they are recombined). Comparing this to the output of the WordLevel tokenizer on the same word in Table 14, the WordPiece algorithm effectively addresses the OOV problem for words similar to those in the training corpus and, according to the output of the WordPiece tokenizer in Table 18, for words unlike those in the training corpus.

Tokenization stage	Representation
Input string:	“you you’re you’ve”
Tokenized string:	[“you”, “you”, “’”, “re”, “you”, “’”, “ve”]
Integral mapping:	[56, 56, 5, 71, 56, 5, 72]

Table 16: Example of WordPiece tokenization of words with shared roots and differing suffixes.

Tokenization stage	Representation
Input string:	“you’ll”
Tokenized string:	[“you”, “’”, “l”, “##l”]
Integral mapping:	[56, 5, 17, 54]

Table 17: WordPiece tokenization of an unknown word similar to some of those in the training data.

Tokenization stage	Representation
Input string:	“anything”
Tokenized string:	[“an”, “##y”, “##t”, “##h”, “##i”, “##n”, “##g”]
Integral mapping:	[61, 49, 37, 53, 32, 36, 39]

Table 18: WordPiece tokenization of a word that does not appear in the training data and is not similar to any of the words in the tokenizer vocabulary.

6.5.3 Byte-Pair Encoding

The Byte-Pair Encoding (BPE) tokenizer algorithm was adapted from a data compression algorithm by Phillip Gage in 1994 [69]. The data compression algorithm works by replacing common pairs of bytes in data with an unused byte to reduce the overall size of the data. The tokenization algorithm works under the same principle, merging frequent pairs of characters or tokens into one token [21].

The vocabulary of the tokenizer is initialized to the base set of characters (letters, numbers, punctuation, etc.). The training procedure begins by counting all pairs (“A”, “B”) of characters that appear in the training data and combines the most frequently occurring pair into one token (“A”, “B” $\rightarrow$ “AB”). This process repeats until a specified vocabulary size has been reached (or a specified number of merge operations have occurred; the final vocabulary size is equal to the base character set plus the number of merge operations).

Algorithm 2 BPE training algorithm implementation in Python. Modified from Sennrich et al. [21].

```
import re, collections

def get_stats(vocab):
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols)-1):
            pairs[symbols[i], symbols[i+1]] += freq
        return pairs
def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
    for word in v_in:
        w_out = p.sub(''.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out

vocab = {...}
num_merges = 10
for i in range(num_merges):
    pairs = get_stats(vocab)
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
    print(best)
```

Tokenization stage	Representation
Input string:	“the quick brown fox jumps over the lazy dog”
Tokenized string:	[“the”, “quick”, “brown”, “fox”, “jumps”, “over”, “the”, “lazy”, “dog”]
Integral mapping:	[36, 70, 72, 66, 74, 69, 36, 68, 64]

Table 19: The Byte-Pair Encoding tokenizer output for a string that appears in the training data.

Algorithm 2 shows a minimal Python implementation for the BPE training procedure.

Finally, comparing the BPE tokenization process with the WordLevel and WordPiece algorithms, we can see that the outputs of the algorithm are nearly identical when the sentences being tokenized have words that appear in the training data (i.e. the BPE outputs in tables 19 and 20 are almost identical to the WordPiece output in tables 15 and 16 except the numerical representation of the tokens) and very similar to that of the WordLevel tokenizer.

Tokenization stage	Representation
Input string:	“you you’re you’ve”
Tokenized string:	[“you”, “you”, “’”, “re”, “you”, “’”, “ve”]
Integral mapping:	[34, 34, 5, 33, 34, 5, 37]

Table 20: Byte-Pair Encoding tokenizer output for syntactically similar words with shared roots.

Tokenization stage	Representation
Input string:	“you’ll”
Tokenized string:	[“you”, “’”, “l”, “l”]
Integral mapping:	[34, 5, 17, 17]

Table 21: Byte-Pair Encoding tokenizer output for a word that did not appear in the training data, but is similar to some of the words in the training data.

Tokenization stage	Representation
Input string:	“anything”
Tokenized string:	[“an”, “y”, “t”, “h”, “i”, “n”, “g”]
Integral mapping:	[39, 30, 25, 13, 14, 19, 12]

Table 22: Byte-Pair Encoding tokenizer output for a word that did not appear in the training data and is not similar to any words in the training data.

When testing the BPE algorithm on words that did not appear in the training data, it is clear that the BPE algorithm effectively deals with and processes OOV words (tables 21 and 22), even when no subunits of the word appear in the training data (Table 22).

Byte-Level Byte-Pair Encoding. This is a subset of the BPE algorithm that is also commonly used for NLP tasks such as neural machine translation, language modeling, and generative tasks (GPT-2 uses a byte-level BPE tokenizer [70]). As Huggingface’s `tokenizers` library implements it, this is a preprocessing step for the tokenizer model that maps all bytes in a string/sentence to their own unique and visible character. Functionally, byte-level BPE is nearly identical to the standard BPE algorithm, described above, except that the beginning of word symbol is rendered and represented by a visible character in the tokenized sequence (see Table 23).

Tokenization stage	Representation
Input string:	“anything”
Tokenized string:	[“Ġa”, “n”, “y”, “t”, “h”, “i”, “n”, “g”]
Integral mapping:	[36, 19, 30, 25, 13, 14, 19, 12]

Table 23: Example of the output of a byte-level Byte-Pair Encoding tokenizer with the begging of a word represented with the “Ġ” character.

6.6 Experiment Results

This section will layout the experiment setup and results for the language modeling pretraining experiments. The language models are divided into two categories, with two models in each category. The first category of models are those trained from scratch meaning that the models weights are initialized according to a truncated normal distribution [71]. The second category of models are based on pretrained checkpoints from BERT and RoBERTa [11,12]. Each category contains a BERT and RoBERTa model.

Considering the success of deep learning models with transformer-based architectures for language modeling and natural language processing (NLP) tasks, more generally, in the conversational and written English domains, it follows that transformer architectures should see widespread use in aviation English for domain-specific NLP tasks. However, the literature shows transformer-based language models have seen sparse use. The most notable example of using a transformer neural network is the application of BERT for knowledge extraction from notice to airmen (NOTAM) messages [28]. While both BERT and RoBERTa have demonstrated that they perform well on English in general [11, 12], it is expected that the pretrained models will have limited performance due to the highly technical and domain-specific nature of aviation English as well as the differing forms of the language; transcribed English speech as opposed to written English. We expect the domain-specific terms and pronunciations will lead to a sufficiently different vocabulary from that of written English such that the models trained from scratch will achieve better performance than the pretrained models.

A standard masked language modeling training procedure is used to pretrain the language models, as described in Section 6.3. Tokens are masked at random as described in the MLM process in Section 6.3. Input sequences are padded to the maximum length the models accept with a special `[PAD]` token and truncated to the maximum length, if greater. The maximum sequence length for BERT and RoBERTa models is 512 tokens [11, 12]. Based on the data analysis in Section 4.7, there are few, if any, samples that meet or exceed the maximum sequence length for BERT and RoBERTa models.

6.6.1 Performance Metrics

The following metrics are defined and used to track the performance of the language models over the course of the training procedures. They also serve as a means of measuring and comparing the performance of the models over the course of training and on the test partition of the corpus.

Cross-Entropy Loss. As mentioned previously, perplexity and cross-entropy loss (often shortened to loss) are used to track the model's performance on the training and validation sets per epoch while training. Cross-entropy loss is defined below.

The cross-entropy loss between two probability distributions, p and q, is typically defined as

$$H(p, q) = -\sum_{x \in X} p(x) \log q(x) \tag{16}$$

where x is a single sample in the discrete distribution X. However, the way in which cross-entropy loss is measured can differ depending on application. PyTorch [66], for example, defines and measures cross-entropy loss as follows:

Let $\mathcal{L}(x, y)$ be the cross-entropy loss function that outputs a $N \times 1$ vector, L, that contains the loss values for each sample in a batch of N samples.

$$\mathcal{L}(x, y) = L = \{l_1, l_2, ..., l_N\}^T \tag{17}$$

Where x is a $512 \times N$ vector representing model inputs (a sequence of tokens) and y is a $1 \times N$ vector representing the target/label for each sample in the batch. The cross-entropy loss for a single sample is then calculated as

$$l_n = -\log \left[\frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})} \right] \quad \forall n = 1..N \tag{18}$$

where l_n is the loss for an individual sample in the batch, and C is the number of class labels. For language modeling, the set of class labels is equivalent to the vocabulary of the tokenizer (and thus the language model). Numerical labels will correspond to token IDs. The loss for a batch is calculated as the mean of the loss for each sample in the batch.

$$\mathcal{L}(x, y) = \frac{1}{N} \sum_{i=1}^{N} l_i \tag{19}$$

Equation (19) can then be extended to find the mean cross-entropy loss over an epoch by swapping l_i for batch means and $\mathcal{L}(x, y)$ for the epoch mean. N would then represent the number of batches in an epoch rather than the number of samples in a batch.

In this case, it is important to note that cross-entropy loss is, basically, a measure of the distance between a models prediction, $\hat{y}$, and the intended target for the prediction, y.

Perplexity. This metric measures a model's ability to predict the most probable token in a sequence. It will measure and track how well the language models can predict the most probable token for a specific point in a sequence. Perplexity is typically defined as

$$\text{PPL} = 2^{-\frac{1}{N} \sum_{i=1}^{N} \log_2 q(x_i)} \tag{20}$$

where N is the number of samples in a sequence, x is an individual sample, and q is the probability model (a language model, for the purposes of this thesis). This is, essentially, the inverse of the probability of a sample appearing. A helpful way to view this is as how "surprised" a probability model is to see a sample, thus the lower the probability of a sample, the higher the perplexity will be.

As with cross-entropy loss, the measure of perplexity can differ between implementations. PyTorch's [66] implementation for perplexity is defined as follows:

Let P and $\hat{Y}$ be vectors representing the perplexity for samples in a batch and non-normalized output neurons in a neural network, respectively.

$$\begin{aligned} P &= \{p_1, p_2, ..., p_N\}^T \\ \hat{Y} &= \{\hat{y}_1, \hat{y}_2, ..., \hat{y}_C\}^T \end{aligned} \tag{21}$$

Where N is the number of samples in a batch and C is the number of class labels. Then, the perplexity for each sample in a batch can be calculated as

$$p_n = \exp\left[\frac{1}{T}\sum_{i=1}^{C} -\log(\hat{y}_i)\right] \quad \forall n = 1..N \tag{22}$$

where p_n is the perplexity for an individual sample in the batch, and T is the number of tokens in the sample. The perplexity for the batch is then calculated as the mean of the perplexities in the batch.

$$\text{PPL} = \frac{1}{N}\sum_{i=1}^{N} p_i \tag{23}$$

Where PPL is the mean perplexity over a batch with N samples. This equation can then be extended to find the mean perplexity over each batch in the epoch, swapping p_i for batch means and PPL for the epoch mean. N would then represent the number of batches in an epoch rather than the number of samples in the batch.

6.6.2 Optimizer Settings

The models are trained with the *AdamW* optimizer [72] using the hyperparameters in Table 24 based on the most effective pretraining configuration presented in [11] and [12]. These settings are kept constant across language model pretraining experiments.

Hyperparameter	Value
β_1	0.9
β_2	0.98
ϵ	1×10^{-6}
Learning Rate	4×10^{-5}
Weight Decay	0.01

Table 24: *AdamW* hyperparameters for training BERT and RoBERTa models.

Corpus split	Split size (%)	Samples
Train	68.99	34,470
Validation	6.999	3,830
Test	30.01	16,415

Table 25: Number of samples and percentage of overall corpus by split.

6.6.3 Data Splits

A train/test split of 70% and 30% and a validation split of 10% of the training data was used for each model. A breakdown of the number of samples in each split is shown in Table 25. The samples present in each split are kept consistent across language model experiments so that the model performances are being compared based on identical data.

6.6.4 Results for the Models Trained from Scratch

The models trained from scratch were trained for 200 epochs. This value was found through two rounds of trial and error by monitoring the curves of the loss and perplexity on the training and validation partitions of the corpus. This value was chosen to ensure that the pretraining procedure did not have an overall negative effect on the model performance. Batch sizes of 16 were used over the course of training; this was the maximum batch size that could be used with the hardware available[13].

A graph of the average loss per step of scratch models on the training partition are shown in Figure 8. Both models quickly converged to low loss values and maintained a relatively

[13] Values larger than 16 would sometimes cause Out of Memory errors on the GPU device used for training, so 16 was used to ensure all models trained continuously without error or interruption. Models were trained on a workstation with 64 GB of RAM and an NVIDIA RTX 3090 GPU with 24 GB of VRAM.

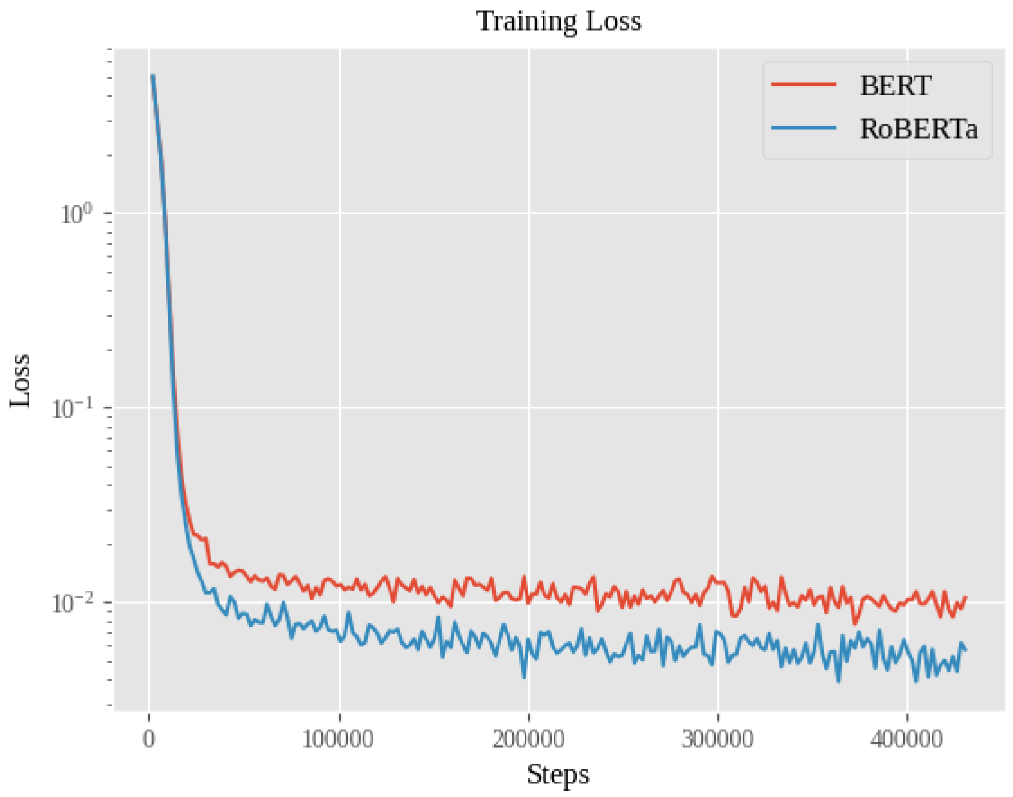

Figure 8: Cross-entropy loss of the models trained from scratch on the training partition of the corpus over time (steps).

stable loss curve for the duration of the training procedure. This suggests that the models are converging on either local or global minima.

The graph of the perplexity of the models on the training partition of the corpus is shown in Figure 9. The perplexity of the models over time is erratic and rarely settles to a consistent trend over the duration of the training procedure. Also note that the perplexity of the models is quite high throughout the training process. This is likely due to an issue with the models rapidly overfitting to the training data as evidenced by graphs of the loss and perplexity of the models on the validation partition of the corpus in Figures 10 and 11. This problem of rapid overfitting and consistently high perplexity is present in the results of the pretrained models as well (see Section 6.6.5). Theories as to why the models' perplexities are so consistently high and why the models are overfitting so quickly will be explored in

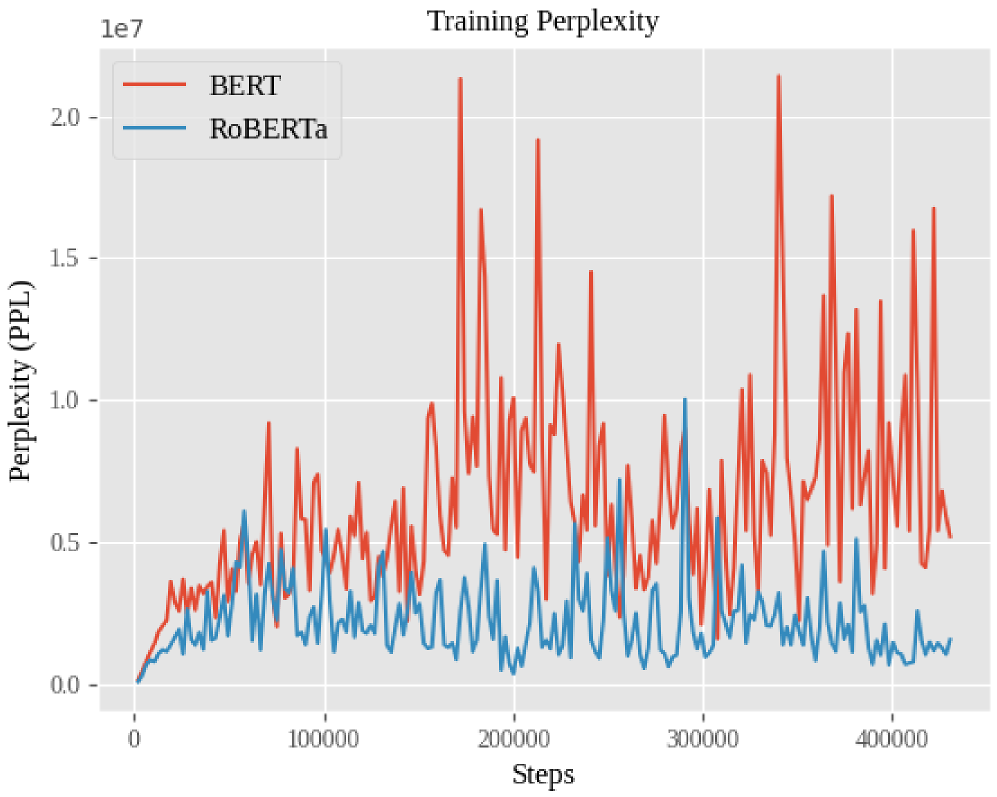

Figure 9: Perplexity of the models trained from scratch on the training partition of the corpus over time (steps).

Section 6.6.6.

6.6.5 Results for the Pretrained Models

The pretrained models were each trained for 100 epochs. As with the models trained from scratch, this value was found through two rounds of trial and error by monitoring the loss and perplexity values of the models on the training and validation partitions of the corpus to find a value that would ensure (or at least minimize) the net negative effects of overfitting. The batch size used for during training for the pretrained models is also 16 (for the same reason as with the models trained from scratch).

The graph of the training loss over time for the pretrained models can be seen in Figure 12. Similarly to the models trained from scratch, the pretrained models quickly converge

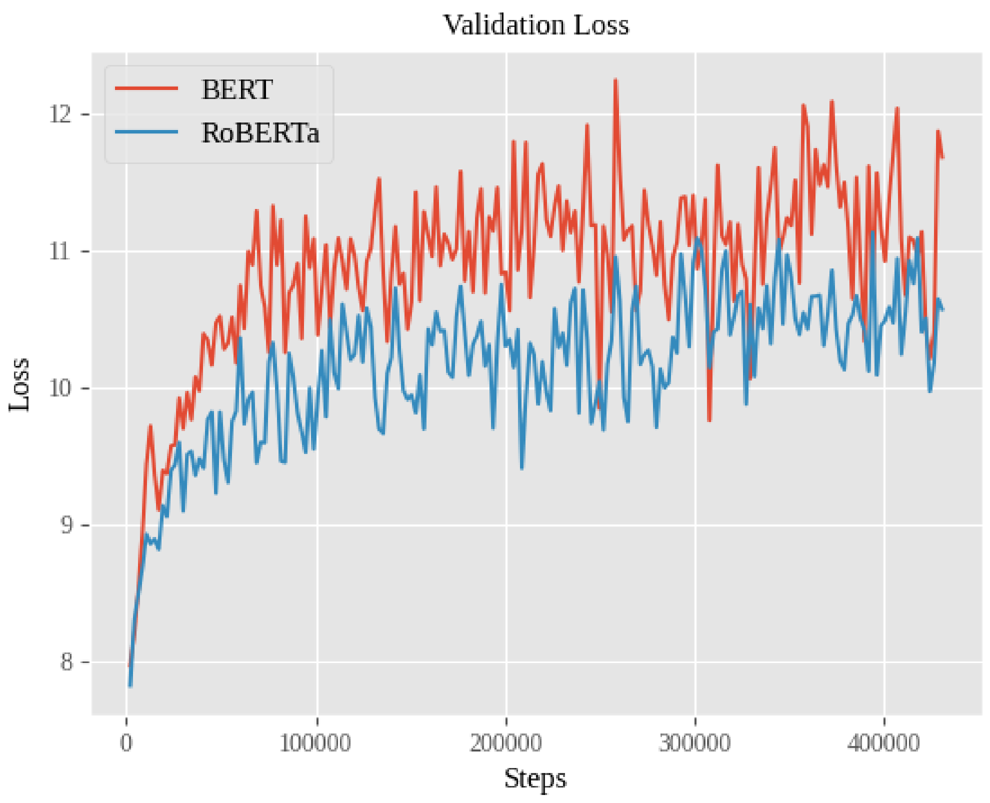

Figure 10: Cross-entropy loss of the models trained from scratch on the validation partition of the corpus over time (steps).

to a consistent low loss value over the duration of the training procedure with little to no deviation. The perplexity of the pretrained models on the training partition over the course of training, however, is much more stable and exhibits a somewhat consistent trend. There are still large spikes in the value of the perplexity throughout the training procedure and the overall values of the perplexity per step are still extremely high (note the scale of 10^9 on the y-axis). As mentioned in Section 6.6.4, this is likely due to overfitting of the models to the data, evidenced by the rapid overfitting shown in Figures 14 and 15. A potential explanation as to why this phenomenon occurs is proposed and explored in Section 6.6.6.

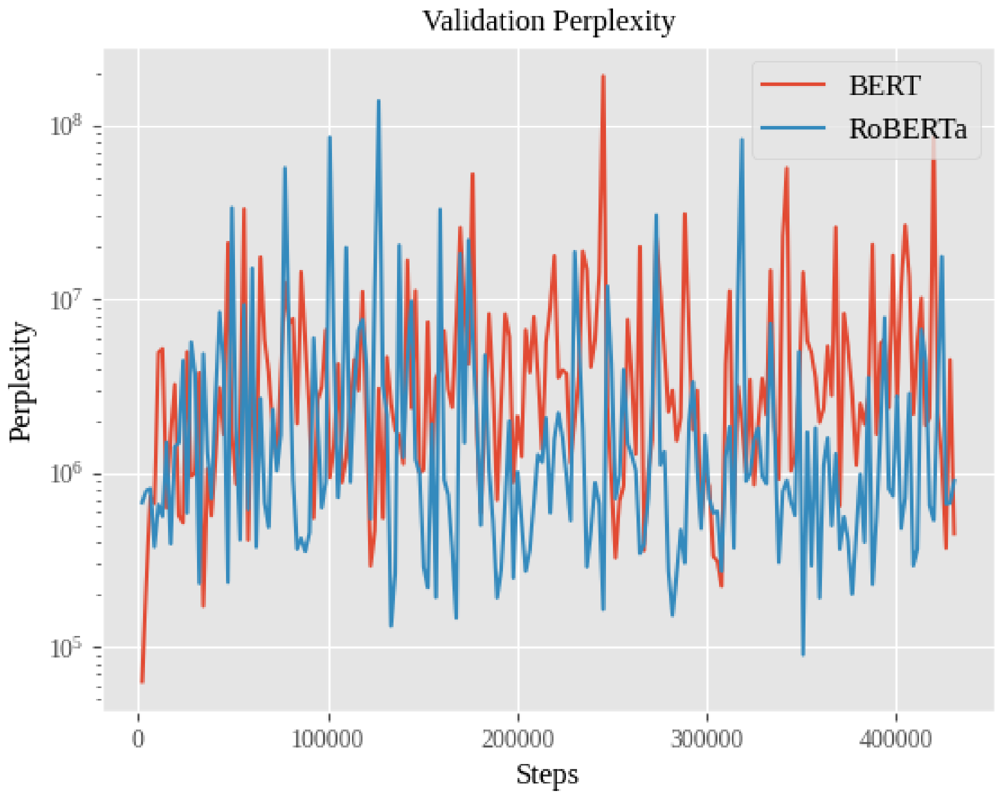

Figure 11: Perplexity of the models trained from scratch on the validation partition of the corpus over time (steps).

6.6.6 Comparison of Results

Both classes of models (scratch and pretrained) quickly reach a steady state in terms of loss on the training partition of the corpus, although the initial loss values are considerably different. Notably, the pretrained models start with a considerably lower loss, about 1.3, compared to the models trained from scratch, which began with a loss value of just over 5.0. This indicates that the models are training well and starting to converge on local or global minima. Although, as the graphs of the loss on the validation partition show, the models are quickly overfitting to the training data, causing the models to begin performing poorly the validation data, which is not seen by the model during training.

Based on the trends exhibited by the cross-entropy loss of the models on the training and validation data, we would expect to see similar trends with the perplexity on the training and

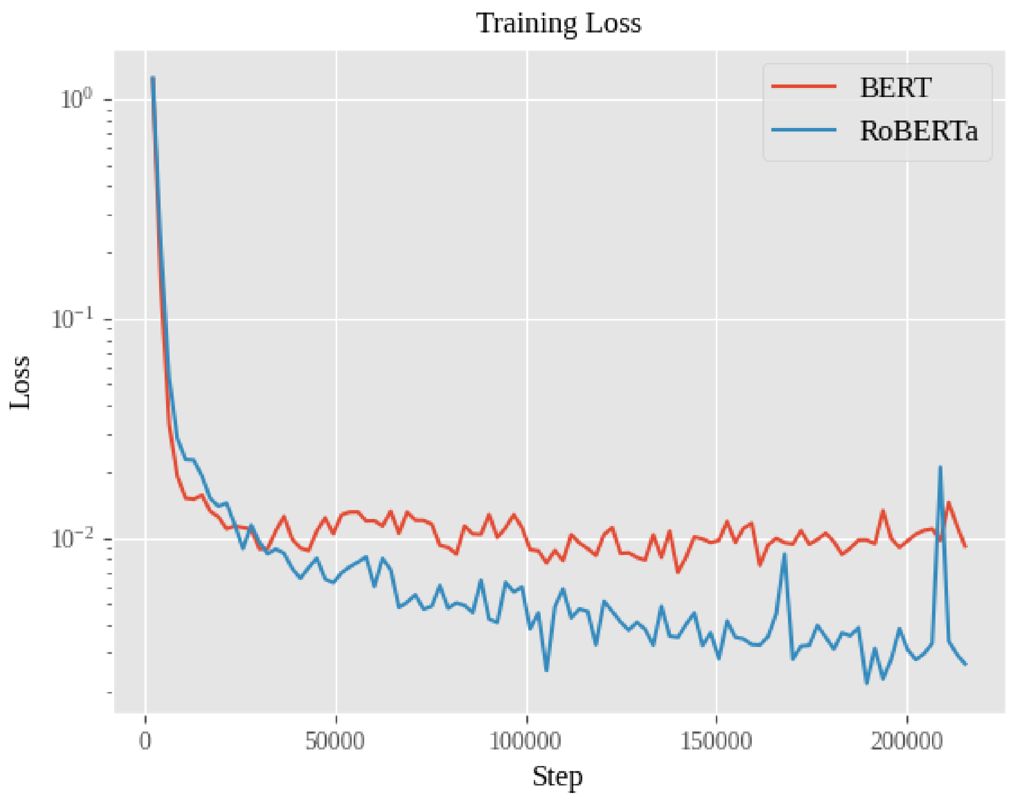

Figure 12: Cross-entropy loss of the pretrained models on the training partition of the corpus over time (steps).

validation partitions. However, the perplexity of the models on the training and validation data yields interesting and somewhat unexpected results. Although the pretrained models have a more stable curve over time, the models trained from scratch result in a lower perplexity. The final perplexity values (over the training set) of the models trained from scratch were markedly better than that of the pretrained models. At the end of training, the BERT and RoBERTa models trained from scratch had perplexity values of 0.52×10^7 and 0.16×10^7, respectively, whereas the pretrained models reached perplexity values of 0.29×10^9 and 0.17×10^9, for BERT and RoBERTa, respectively. Using the following equation

$$\text{RD} = \frac{v - v_{ref}}{v_{ref}} \tag{24}$$

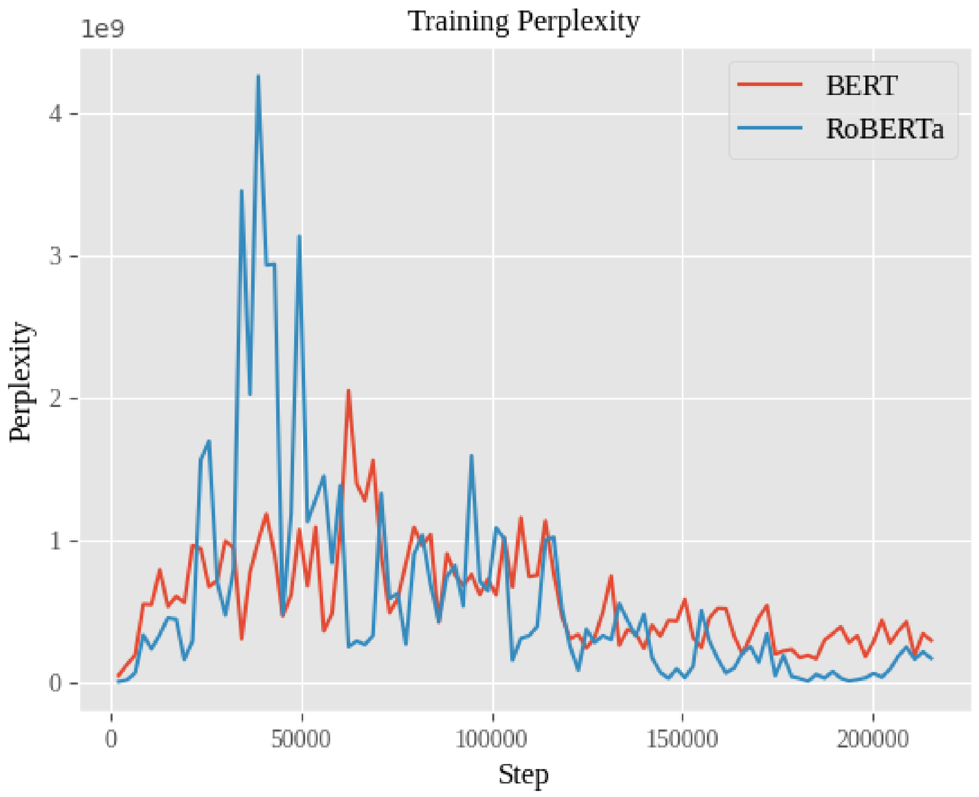

Figure 13: Perplexity of the pretrained models on the training partition of the corpus over time (steps).

where v is a tested value and v_{ref} is a reference value. The relative difference in perplexity between the scratch and pretrained models on the final training epoch is 55.24 for the BERT models, and 104.57, for the RoBERTa models. Using the following equation for absolute difference,

$$\text{AD} = |a - b| \tag{25}$$

this equates to an absolute difference of 28.74×10^7 and 16.43×10^7 for the BERT and RoBERTa models, respectively.

The results from the training procedure show that the models trained from scratch outperformed their pretrained counterparts. This initially indicates that pretraining models on in-domain data may achieve the best language model performance, then continuing to

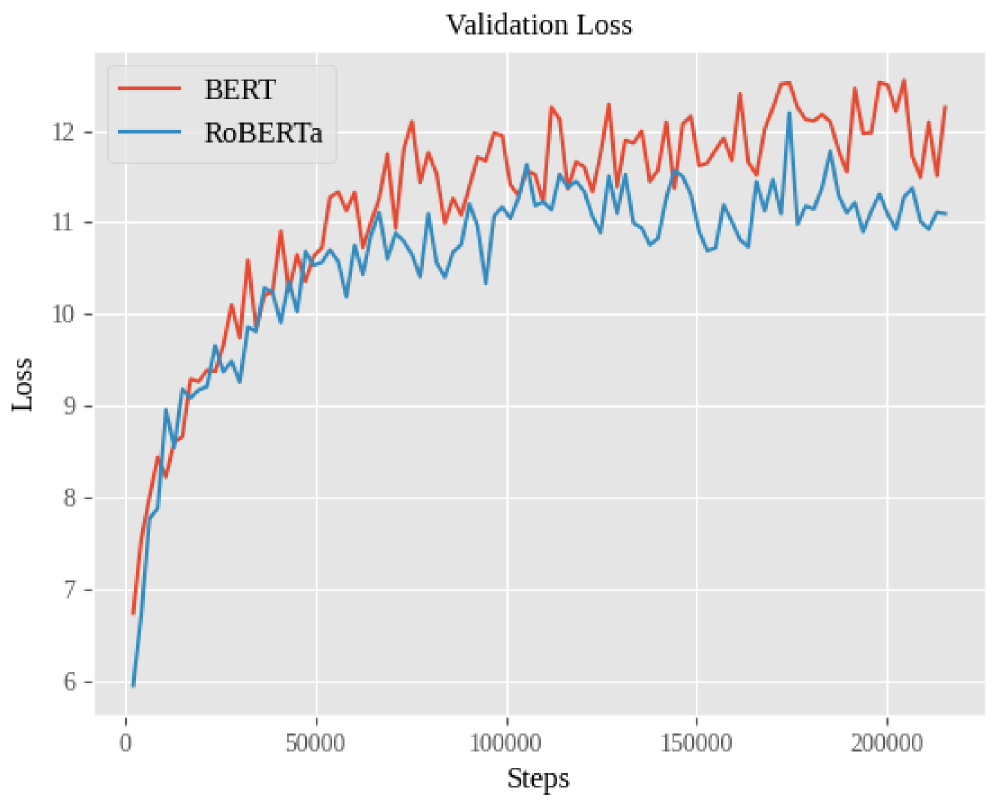

Figure 14: Cross-entropy loss of the pretrained models on the validation partition of the corpus over time (steps).

fine-tune on specific downstream tasks. This seems to be a direct contradiction of the current recommended practice of using one of the pretrained checkpoints from Devlin et al. [11] and Liu et al. [12] then fine-tuning on specific downstream tasks. However, this is only partially upheld by the results of the model performance on the test partition of the corpus (Table 26). For the BERT models, the pattern exhibited during training is consistent with the results on the test partition, resulting in a relative difference of 8.62 and an absolute difference of 1.24×10^7 (using equations (25) and (24), respectively). For the RoBERTa models, however, the relative difference in perplexity is -0.34 with an absolute difference of 0.36×10^7. These results show that the pretrained RoBERTa model performed better during pretraining than the corresponding model that was trained from scratch, unlike the BERT models. The loss of each model over the test set is somewhat high, but they are around the

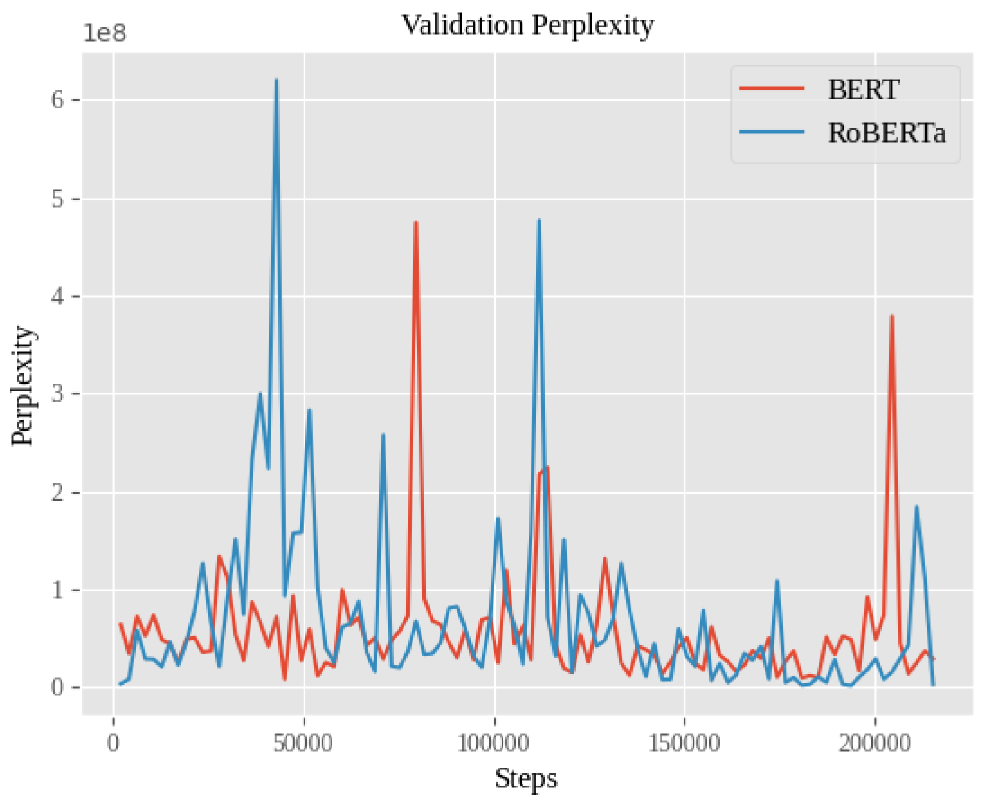

Figure 15: Perplexity of the pretrained models on the validation partition of the corpus over time (steps).

same values for each model, which is consistent with the loss statistics during training. A higher loss value on unseen (test) data is expected.

The BERT model trained from scratch had the best performance among the tested models followed closely by the pretrained RoBERTa model. The pretrained BERT model used both MLM and NSP pretraining objectives to create the checkpoint used in this thesis. The pretrained RoBERTa model, on the other hand, used only the MLM pretraining objective. It is readily apparent from the results of [12] that RoBERTa's modified pretraining methodology resulted in better model performance for in-domain problems. However, it seems that the pretraining method also allows RoBERTa to generalize better than BERT when transfer learned to other NLP domains. It is conceivable that, given additional training data in the future, the pretrained RoBERTa model could surpass the performance of the

Model	Loss	Perplexity ($\times 10^7$)
BERT (scratch)	10.54	0.14
RoBERTa (scratch)	11.41	1.04
BERT (pretrained)	11.00	1.38
RoBERTa (pretrained)	11.80	0.68

Table 26: Results on the test set of data after pretraining the BERT and RoBERTa models from scratch and pretrained checkpoints. Loss and perplexity values are averaged over the test partition of the corpus. For both perplexity and loss; lower values are better.

BERT model trained from scratch. However, with the current volume of data available in this domain, it is clear from the models' results on the corpus test partition that a BERT model trained from scratch will still be the best option for aviation English NLP tasks.

With the results and performance of the models established relative to each other, the high perplexity of each model must be addressed. While the loss values converge nicely during training, the perplexity is somewhat erratic and consistently high, rarely converging to a consistent curve. However, it exhibits a clear trend as training progresses. Adjusting the hyperparameters of the optimizer did not solve this issue, and changing the hyperparameters of the model (e.g. sequence length, number of hidden layers, attention heads, etc.) was not desirable for performing a direct comparison between the pretrained checkpoints and the base models to study the effectiveness of the models on orthographic transcriptions of aviation English. The validation curves for each model (Figures 10 and 14) clearly suggest that the language models are rapidly overfitting to the training data, which is what is causing the loss on the validation partition to increase as the models are trained further. It is difficult to determine what exactly is causing this phenomenon of the models rapidly and extremely overfitting to the training data, so several theories were developed in an attempt to explain it (some of which will be elaborated upon in Section 8):

Small volume of training data. Especially when compared to *Attention is All You Need*, BERT, and RoBERTa, [11, 12, 22] the number of samples used in the training data dwarfs that of the state-of-the-art. It has been exhaustively demonstrated that neural networks perform better with large volumes of data. The inception of RoBERTa itself is based on the idea that BERT was not trained on enough data [12]. Increasing the volume of in-domain data used for pretraining (and training) would be the most effective way to immediately improve the performance of the models.

Different forms of English. The BERT and RoBERTa pretraining checkpoints were created by training the models on literary works and typed articles such as those from *Wikipedia*. The data used in this work are orthographic transcriptions of aviation English, and while the two data sources share the common root of English, transfer learning from the casual style of written English to the technical, highly specific use of aviation English yields limited results, as evidenced above, especially with relatively small amounts of data.

Mixed regional data sources. As described in Section 4, the data aggregated for use here comes from several different regions, namely international airports in the US [47] and international and domestic airports in Europe [30, 33, 48]. During data processing, it was ensured that only English transcriptions were included in the corpus. However, proper nouns from each region will be present in the data. Additionally, dialectical differences and varying levels of English proficiency will affect the structure of token sequences across regions. All of these combined facets may contribute to the high perplexity of the language models relative to other works. It should be noted that the varied regional data sources are not necessarily bad. Depending on the application and desired robustness of the model, the ability to interpret international sources and contexts can be desirable for model selection; however, it will require significantly more data and proportionally balanced regional contexts within the corpus for the pretraining process to be effective for downstream tasks.

Up to this point, we have followed a procedure for studying the effectiveness of the models that is somewhat symmetrical to that of BERT and RoBERTa, so at this point, the models should be trained for downstream tasks to measure the effectiveness of the pretraining procedures further. In the aviation domain and for this particular data, that would mean training for tasks such as callsign identification, speaker identification, flight phase classification, clearance recognition, etc. However, labels for these tasks are not present in the corpora used here. Corpora with relevant labels for downstream tasks could not be obtained within an appropriate time to run and include the experiments in this work. Some potential algorithms were developed to mine labels from the corpora aggregated for this work, which will be expanded upon in section 8. For these reasons, testing model performance and effectiveness on downstream tasks must be left for future work.

7 Conclusions

Four aviation English corpora are aggregated and unified for use in neural language modeling and speech recognition. A masked language modeling pretraining methodology is then used to pretrain two classes of BERT and RoBERTa language models (models trained from scratch and pretrained models transfer learned to aviation English). For ASR we used MFCCs, calculated from the audio signals aggregated from the aviation English corpora, and the transcripts used for language modeling to train and evaluate the models.

Contrary to the recommended best practice of using pretrained checkpoints and training for downstream tasks, we find that, in the domain of aviation English, models achieve better performance when pretrained from scratch than model checkpoints transfer learned from the available BERT and RoBERTa pretrained checkpoints. Although the pretrained RoBERTa model achieved better performance on the test partition of the corpus than expected, the pretraining methods laid out in this thesis resulted in the BERT model trained from scratch achieving the lowest loss and perplexity overall. This trend also emerges in the ASR models; the Jasper model trained from scratch achieved the best performance among the models evaluated, including those from pre-existing checkpoints.

The results from the experiments in this work indicate that models trained from scratch for natural language processing tasks with limited data availability in the aviation domain are more effective than transfer learning pretrained models. Thus, models trained from scratch should be fine-tuned and used for downstream tasks over pretrained models. The most significant limiting factor to the performance of the models is the small volume of aviation English data available for pretraining and downstream task use. Investment in creating expansive, high-quality, openly available aviation English corpora with appropriate formatting for developing machine learning applications is also highly recommended.

Based on the results from training both ASR and language models from scratch and comparing the results with those fine-tuned from existing checkpoints, it is clear that, in both cases, the models trained from scratch not only have better performance overall but will also have a higher likelihood of reliable callsign detection/identification performance due to the higher individual performance overall. For future applications of NLP tasks in the aviation English domain, we recommend pretraining models from their base initializations using in-domain data and fine-tuning on additional labeled in-domain data for downstream tasks.

8 Future Work

This section is intended to serve as a brief proposal or set of suggestions for future work based on the results of this work. The main focus of these suggestions is corpus-specific algorithms and data augmentation techniques to enable testing and experiments to determine the effectiveness of the pretraining procedures covered earlier in this thesis.

8.1 Next Sentence Prediction

This training procedure was introduced by Devlin et al. [11] as a pretraining procedure for BERT and is described in more detail in Section 6.3. This can be used on top of masked language modeling for language model pretraining. However, it requires a knowledge of which sequences in the corpus appear next to each other. For pretraining transformer-based neural language models for the aviation domain (with orthographic transcriptions), this same procedure can be used to classify whether two transmissions occur in the correct order. Generalizing a bit, we can consider two individual transmissions to be A and B; the training objective for the network is then to predict whether transmission B occurs in response to transmission A. This would be considered a binary classification task; B either occurs in response to A or it does not. In practice, for a true label (meaning B does, in fact, occur in response to A), transmission A, for example, can be a pilot request to ATC and transmission B would be ATC's response. A false label (meaning B does not occur in response to A) would be made up of transmissions randomly shuffled such that A and B do not occur together in the corpus. Note that, since these are only two transmissions, neither A nor B will necessarily be one side of the transmission i.e. A can be either a pilot or ATC. Of the four corpora used in this work, ATCC is the only corpus that can be mined for an NSP-like pretraining procedure since the raw transcripts have indicators for time (start and end indicators), transmitting party, and intended receiver. With this information, we can estimate the correct order of transmissions between two parties to a reasonable degree of certainty (manually or programmatically). For example, using the following transmissions from ATCC corpus, in which the Boston-Logan tower is communicating with PAA540. The transmission below can be used as A:

> CLIPPER SIXTY FIVE FORTY IS AH FOUR FROM RIPIT CLEARED FOR THE I L S D M E TWO SEVEN APPROACH

and the response from PAA540 to the Boston-Logan tower can be used as B:

> AH ROGER CLEARED FOR THE APPROACH AH CLIPPER FIVE FORTY THANK YOU

In this case, A and B occur in the middle of an interaction. There are two transmissions between PAA540 and the Boston-Logan tower that occur prior to the two above. We can consider the two instances of A and B above as a true label, meaning they occur one after the other in the corpus. To construct a false label, we can simply reverse the input of A and

B, i.e. A becomes B and B becomes A, or choose another transmission at random from the corpus to replace either A or B. For proper construction of the corpus for the NSP pretraining task, the distribution of true and false labels should be evenly balanced. If there are 500 samples labeled as true, then there should be 500 samples labeled as false.

8.2 Callsign Detection/Identification

This is the process of identifying an aircraft callsign from the transcription of the spoken callsign. This is a popular subtask for automatic speech recognition applications in aviation, but postprocessing speech recognition outputs can also be accomplished through a language model. ATCC is a good candidate corpus for this task. There are two potential methods for implementing this as a downstream task: (1) In transmissions from air-traffic controllers to specific aircraft, use the receiver (the "TO" label in ATCC's raw transcriptions) as the label for each sequence or (2) Identify the sequence of tokens, in each transmission, that correspond to the callsign of the aircraft and use those tokens as the labels for each sequence. Each of these methods brings its problems. The first method effectively creates an ever-expanding classification problem with infinitely many labels, so a tokenization or decoding scheme must be created to address this. The second method requires manually labeling every sequence in the corpus, which may not be feasible in all cases. Lastly, both methods may lack sufficient context to predict aircraft callsigns correctly. Callsigns are agreed upon between air carriers and air traffic control, but the callsign used will not always correspond directly to the air carrier's official three-letter designation for flights. For example, United Airlines uses the three-letter designator "UAL", however, the agreed upon spoken, and therefore transcribed, callsign is "united", so the flight "UAL774" will use the callsign "united seven seven four"[14]. General aviation (GA) callsigns introduce more variability and do not use previously agreed upon callsigns. Instead, for general aviation (GA) aircraft, the

[14]The FAA maintains a list of companies, callsigns, and three-letter designators at https://www.faa.gov/air_traffic/publications/atpubs/cnt_html/chap3_section_4.html

spoken information is based on the registration or tail number of the aircraft, but not every element of the registration number is necessarily required information in communications. For example, the FAA does not require the make or region of registration of the aircraft in spoken communications, however, they can and do appear in communications. The following transcript is taken from ATCC and is a communication from ATC to an aircraft with the registration number "N01C":

> TWIN CESSNA ZERO ONE CHARLIE TURN RIGHT HEADING ZERO NINER ZERO

Note that the make of the aircraft, though not required, is included and the region of registration (the *N* at the beginning of the registration number) is excluded. The callsign that should be identified in this case is "TWIN CESSNA ZERO ONE CHARLIE" and, ideally, this should be mapped to the registration "N01C", although this is difficult without additional context such as the region of airport and make of the aircraft. Even with this information, it does not guarantee correct identification of the aircraft; only an increased likelihood. This variability of information present in callsigns highlights the difficulty of callsign identification tasks. A well-trained language model with post-processing rules to map callsigns to three-letter airline designators could achieve exceptionally high performance on this task.

8.3 Speaker (Role) Identification

Generally, speaker identification is a task to classify specific speakers in a series of communications. This subtask usually falls under automatic speech recognition, but it is also feasible for language models under some circumstances. In aviation, this could be as simple as identifying the side of the communication, i.e., an air-traffic controller or a pilot, or it can be complex enough to be grouped under callsign identification tasks. In the simplest form, this can be accomplished easily using ATCC since controller positions are labeled along with transmitting/receiving aircraft.

9 Acknowledgements

The following libraries were used to develop, train, test, and validate the models used in this work as well as perform data processing and analysis:

- PyTorch [66]
- HuggingFace transformers (and tokenizers) [68]
- PyTorch Lightning [73]
- Sci-Kit Learn [61]

10 Appendix A: Examples of Samples Detected as Outliers

The outlying sample that was removed from the corpus is below[15]:

> uh uh have a nice evening thank you bye bye ciao yes to the irish pub you know it uh you have to ask i have no idea where it is i'm just walking behind the others uh no uh but you come in front and you ask and he will tell you but i really don't know i don't even know the address okay it's in paris definitely okay bye

This sample has a sequence length of 71 tokens, well above the mean of the aggregated corpora. The decision was made to remove this sample because it has little relevance to aviation communications, especially the ones being examined by the model in this thesis.

A few other samples, detected as outliers, but not removed are shown in Table 27.

[15]The preprocessing effects on the string, capitalization, punctuation, etc. are left unmodified for display purposes

Outlying Sample	Sequence Length
ah they're probably putting well i don't know if they are are not the weather is niner hundred scattered measured ceiling one thousand five hundred broken two thousand five hundred overcast visibility one zero temperature five three dew point five zero wind ah zero six zero at three altimeter two niner niner seven	53
american eight forty six descend and maintain one thousand six hundred turn left heading zero three zero join the localizer you're four from oxonn maintain one thousand six hundred till established cleared i l s three six approach	38
t w a seven hundred traffic's a helicopter below you at the pentagon he's no factor there's a cessna at one o'clock three miles three thousand five hundred v f r circling he'll stay east of the river	38
yeah we're supposed to ah the thing is now we're going to get you a way out of line for your profile ah in your turn you can continue right around the left to ah zero four zero and then we're going to have to bring you all the way back around to the right	55
actually continental ten seventy two he's just ahead a little bit to your right now ten seventy two you're five from ripit maintain two thousand til established cleared i l s d m e approach runway two seven	38
continental three eighty two from the i loner cross loner at three or above cleared i l s d m e two seven approach maintain one seven zero knots to loner correction one seven zero knots until ripit	38
u s r sixteen fifteen turn left heading two niner zero three from loner cross loner at three or above cleared i l s d m e runway two seven approach maintain one seven zero knots until ripit	38
clipper sixty five forty seven miles from ripit heading three zero zero maintain three thousand til established on the localizer cleared i l s d m e runway two seven approach speed one ninety or greater to ripit	38
cessna three ten bravo whiskey boston tower ah move up and ah the heavy jet's going to runway two two left you'll ah depart prior to both heavies you can move up to the next avlable taxiway it's just it's about a thousand feet ahead turn right and hold short of runway two two right acknowledge hold	57

Table 27: A few examples of samples detected as outliers that were kept in the aggregated corpus. Samples with sequence lengths significantly higher or lower than the average for the aggregated corpus were detected as outliers and flagged for additional analysis.

References

[1] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep Speech: Scaling up end-to-end speech recognition," Dec. 2014, arXiv:1412.5567 [cs]. [Online]. Available: http://arxiv.org/abs/1412.5567

[2] H. Lee, P. Pham, Y. Largman, and A. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," in *Advances in Neural Information Processing Systems*, vol. 22. Curran Associates, Inc., 2009. [Online]. Available: https://proceedings.neurips.cc/paper/2009/hash/a113c1ecd3cace2237256f4c712f61b5-Abstract.html

[3] J. Li, V. Lavrukhin, B. Ginsburg, R. Leary, O. Kuchaiev, J. M. Cohen, H. Nguyen, and R. T. Gadde, "Jasper: An End-to-End Convolutional Neural Acoustic Model," Aug. 2019, arXiv:1904.03288 [cs, eess]. [Online]. Available: http://arxiv.org/abs/1904.03288

[4] S. Kriman, S. Beliaev, B. Ginsburg, J. Huang, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, and Y. Zhang, "Quartznet: Deep Automatic Speech Recognition with 1D Time-Channel Separable Convolutions," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2020, pp. 6124–6128, iSSN: 2379-190X.

[5] H. Helmke, Y. Oualil, and M. Schulder, "Quantifying the benefits of speech recognition for an air traffic management application," *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung 2017*, pp. 114–121, 2017, publisher: TUDpress, Dresden.

[6] H. Helmke, J. Rataj, T. Mühlhausen, O. Ohneiser, H. Ehr, M. Kleinert, Y. Oualil, M. Schulder, and D. Klakow, "Assistant-based speech recognition for ATM applications," 2015.

[7] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer Sentinel Mixture Models," Sep. 2016. [Online]. Available: https://arxiv.org/abs/1609.07843v1

[8] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, ser. HLT '11. USA: Association for Computational Linguistics, Jun. 2011, pp. 142–150.

[9] B. Paltridge and S. Starfield, Eds., *The Handbook of English for Specific Purposes*. Chichester, West Sussex: Wiley-Blackwell, 2013.

[10] D. Zhang, N. Maslej, E. Brynjolfsson, J. Etchemendy, T. Lyons, J. Manyika, H. Ngo, J. C. Niebles, M. Sellitto, E. Sakhaee, Y. Shoham, J. Clark, and R. Perrault, "The AI Index 2022 Annual Report," May 2022, arXiv:2205.03468 [cs]. [Online]. Available: http://arxiv.org/abs/2205.03468

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," May 2019, arXiv:1810.04805 [cs]. [Online]. Available: http://arxiv.org/abs/1810.04805

[12] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A Robustly Optimized BERT Pretraining Approach," Jul. 2019, arXiv:1907.11692 [cs]. [Online]. Available: http://arxiv.org/abs/1907.11692

[13] T. Pellegrini, J. Farinas, E. Delpech, and F. Lancelot, "The Airbus Air Traffic Control speech recognition 2018 challenge: towards ATC automatic transcription and call

sign detection," in *Interspeech 2019*, Sep. 2019, pp. 2993–2997, arXiv:1810.12614 [cs, eess]. [Online]. Available: http://arxiv.org/abs/1810.12614

[14] E. Delpech, M. Laignelet, C. Pimm, C. Raynal, M. Trzos, A. Arnold, and D. Pronto, "A Real-life, French-accented Corpus of Air Traffic Control Communications," May 2018. [Online]. Available: https://hal.science/hal-01725882

[15] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension," Oct. 2019, arXiv:1910.13461 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1910.13461

[16] S. Yadlowsky, L. Doshi, and N. Tripuraneni, "Pretraining Data Mixtures Enable Narrow Model Selection Capabilities in Transformer Models," Nov. 2023, arXiv:2311.00871 [cs, stat]. [Online]. Available: http://arxiv.org/abs/2311.00871

[17] T. Jech, "Chapter 1 Axiomatic Set Theory," in *Pure and Applied Mathematics*. Elsevier, 1978, vol. 79, pp. 1–77. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0079816908611927

[18] "The American Heritage Dictionary of the English Language," Nov. 2011. [Online]. Available: https://ahdictionary.com/word/search.html?q=word

[19] J. D. Allen, D. Anderson, J. Becker, R. Cook, M. Davis, P. Edberg, M. Everson, A. Freytag, L. Iancu, and R. Ishida, "The unicode standard," *Mountain view, CA*, pp. 660–664, 2012, publisher: Citeseer.

[20] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and

J. Dean, “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation,” Oct. 2016, arXiv:1609.08144 [cs]. [Online]. Available: http://arxiv.org/abs/1609.08144

[21] R. Sennrich, B. Haddow, and A. Birch, “Neural Machine Translation of Rare Words with Subword Units,” Jun. 2016, arXiv:1508.07909 [cs]. [Online]. Available: http://arxiv.org/abs/1508.07909

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention Is All You Need,” Jun. 2017. [Online]. Available: https://arxiv.org/abs/1706.03762v5

[23] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving language understanding by generative pre-training,” 2018, publisher: OpenAI.

[24] M. Schuster and K. Nakajima, “Japanese and Korean voice search,” in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2012, pp. 5149–5152, iSSN: 2379-190X. [Online]. Available: https://ieeexplore.ieee.org/document/6289079

[25] M. Ragnarsdottir, H. Waage, and E. Hvannberg, “Language technology in air traffic control,” in *Digital Avionics Systems Conference, 2003. DASC ’03. The 22nd*, vol. 1, Oct. 2003, pp. 2.E.2–21–13 vol.1.

[26] L. Tanguy, N. Tulechki, A. Urieli, E. Hermann, and C. Raynal, “Natural language processing for aviation safety reports: From classification to interactive analysis,” *Computers in Industry*, vol. 78, pp. 80–95, May 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166361515300464

[27] T. Madeira, R. Melício, D. Valério, and L. Santos, “Machine Learning and Natural Language Processing for Prediction of Human Factors in Aviation

Incident Reports," *Aerospace*, vol. 8, no. 2, p. 47, 2021, num Pages: 47 Place: Basel, Switzerland Publisher: MDPI AG. [Online]. Available: https://www.proquest.com/docview/2524211382/abstract/5B09B07DC1084F4EPQ/1

[28] A. Arnold, F. Ernez, C. Kobus, and M.-C. Martin, "Knowledge extraction from aeronautical messages (NOTAMs) with self-supervised language models for aircraft pilots," in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track*. Hybrid: Seattle, Washington + Online: Association for Computational Linguistics, Jul. 2022, pp. 188–196. [Online]. Available: https://aclanthology.org/2022.naacl-industry.22

[29] D. Guo, J. Zhang, B. Yang, and Y. Lin, "A Comparative Study of Speaker Role Identification in Air Traffic Communication Using Deep Learning Approaches," Aug. 2022, arXiv:2111.02041 [cs, eess]. [Online]. Available: http://arxiv.org/abs/2111.02041

[30] L. Šmídl, J. Švec, D. Tihelka, J. Matoušek, J. Romportl, and P. Ircing, "Air traffic control communication (ATCC) speech corpora and their use for ASR and TTS development," *Language Resources and Evaluation*, vol. 53, no. 3, pp. 449–464, Sep. 2019. [Online]. Available: https://doi.org/10.1007/s10579-019-09449-5

[31] Z.-G. Juan, P. Motlicek, Q. Zhan, R. Braun, and K. Vesely, "Automatic speech recognition benchmark for air-traffic communications." ISCA, 2020, pp. 2297–2301. [Online]. Available: http://infoscience.epfl.ch/record/284987

[32] S. Badrinath and H. Balakrishnan, "Automatic Speech Recognition for Air Traffic Control Communications," *Transportation Research Record*, vol. 2676, no. 1, pp. 798–810, Jan. 2022, publisher: SAGE Publications Inc. [Online]. Available: https://doi.org/10.1177/03611981211036359

[33] K. Hofbauer, S. Petrik, and H. Hering, “The ATCOSIM Corpus of Non-Prompted Clean Air Traffic Control Speech,” in *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC’08)*. Marrakech, Morocco: European Language Resources Association (ELRA), May 2008.

[34] W. Han, Z. Zhang, Y. Zhang, J. Yu, C.-C. Chiu, J. Qin, A. Gulati, R. Pang, and Y. Wu, “ContextNet: Improving Convolutional Neural Networks for Automatic Speech Recognition with Global Context,” May 2020, arXiv:2005.03191 [cs, eess]. [Online]. Available: http://arxiv.org/abs/2005.03191

[35] S. Majumdar, J. Balam, O. Hrinchuk, V. Lavrukhin, V. Noroozi, and B. Ginsburg, “Citrinet: Closing the Gap between Non-Autoregressive and Autoregressive End-to-End Models for Automatic Speech Recognition,” Apr. 2021, arXiv:2104.01721 [eess]. [Online]. Available: http://arxiv.org/abs/2104.01721

[36] J. Zuluaga-Gomez, I. Nigmatulina, A. Prasad, P. Motlicek, K. Veselý, M. Kocour, and I. Szöke, “Contextual Semi-Supervised Learning: An Approach To Leverage Air-Surveillance and Untranscribed ATC Data in ASR Systems,” Aug. 2021, arXiv:2104.03643 [cs, eess]. [Online]. Available: http://arxiv.org/abs/2104.03643

[37] A. Srinivasamurthy, P. Motlicek, I. Himawan, G. Szaszak, Y. Oualil, and H. Helmke, Eds., *Semi-supervised Learning with Semantic Knowledge Extraction for Improved Speech Recognition in Air Traffic Control*, 2017, meeting Name: Proceedings of Interspeech 2017.

[38] K. Money, “Aural Communication in Aviation.” Conference Proceedings, 1981, section: Technical Reports. [Online]. Available: https://apps.dtic.mil/sti/citations/ADA103395

[39] R. North, S. Mountford, and H. Bergeron, “Application of speech recognition and synthesis in the general aviation cockpit,” in *General Aviation Technology*

Conference, ser. General Aviation Technology Conference. American Institute of Aeronautics and Astronautics, Jul. 1984. [Online]. Available: https://arc.aiaa.org/doi/10.2514/6.1984-2239

[40] R. NORTH and H. Bergeron, "Systems concept for speech technology application in general aviation," 1984, p. 2639.

[41] M. S. Mayer and A. W. Lindburg, "Electronic Voice Communications Improvements for Army Aircraft," in *Aural Communications in Aviation*, 1981, pp. 35–46.

[42] J. D. Mosko, ""Clear Speech": A Strategem for Improving Radio Communications and Automatic Speech Recognition in Noise," in *Aural Communications in Aviation*, 1981, pp. 29–32.

[43] J. C. Cotton, M. E. McCauley, R. A. North, and M. I. Strieb, "Development of Speech Input/Output Interfaces for Tactical Aircraft," Tech. Rep., Jul. 1983, section: Technical Reports. [Online]. Available: https://apps.dtic.mil/sti/citations/ADA136485

[44] D. Schäfer, "Context-sensitive speech recognition in the air traffic control simulation," 2000, publisher: Citeseer.

[45] A. Lechner, P. Mattson, and K. Ecker, "Voice recognition: software solutions in real-time ATC workstations," *IEEE Aerospace and Electronic Systems Magazine*, vol. 17, no. 11, pp. 11–16, Nov. 2002, conference Name: IEEE Aerospace and Electronic Systems Magazine.

[46] J. M. Cordero, M. Dorado, and J. M. de Pablo, "Automated speech recognition in ATC environment," 2012, pp. 46–53.

[47] J. J. Godfrey, "Air Traffic Control Complete," 1994. [Online]. Available: https://catalog.ldc.upenn.edu/LDC94S14A

[48] I. Szoke, S. Kesiraju, O. Novotny, M. Kocour, K. Vesely, and J. H. Cernocky, "Detecting English Speech in the Air Traffic Control Voice Communication," Apr. 2021. [Online]. Available: https://arxiv.org/abs/2104.02332v1

[49] J. C. Segura, T. Ehrette, A. Potamianos, D. Fohr, I. Illina, P.-a. Breton, V. Clot, R. Gemello, M. Matassoni, and P. Maragos, "The HIWIRE database, a noisy and non-native English speech corpus for cockpit communication. http://www.hiwire.org," 2007.

[50] L. Graglia, B. Favennec, and A. Arnoux, "Vocalise: assessing the impact of data link technology on the R/T channel," in *24th Digital Avionics Systems Conference*, vol. 1, Oct. 2005, pp. 5.C.2–51, iSSN: 2155-7209.

[51] J. Zuluaga-Gomez, K. Veselý, I. Szöke, A. Blatt, P. Motlicek, M. Kocour, M. Rigault, K. Choukri, A. Prasad, S. S. Sarfjoo, I. Nigmatulina, C. Cevenini, P. Kolčárek, A. Tart, J. Černocký, and D. Klakow, "ATCO2 corpus: A Large-Scale Dataset for Research on Automatic Speech Recognition and Natural Language Understanding of Air Traffic Control Communications," Jun. 2023, arXiv:2211.04054 [cs, eess]. [Online]. Available: http://arxiv.org/abs/2211.04054

[52] A. V. Moere, M. Suzuki, R. Downey, and J. Cheng, "Implementing ICAO language proficiency requirements in the Versant Aviation English Test," *Australian Review of Applied Linguistics*, vol. 32, no. 3, pp. 27.1–27.17, Jan. 2009, publisher: John Benjamins. [Online]. Available: https://www.jbe-platform.com/content/journals/10.2104/aral0927

[53] H. Gürlük, H. Helmke, M. Wies, H. Ehr, M. Kleinert, T. Mühlhausen, K. Muth, and O. Ohneiser, "Assistant based speech recognition - another pair of eyes for the Arrival Manager," in *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, Sep. 2015, pp. 3B6–1–3B6–14, iSSN: 2155-7209.

[54] H. Helmke, M. Kleinert, O. Ohneiser, H. Ehr, and S. Shetty, “Machine Learning of Air Traffic Controller Command Extraction Models for Speech Recognition Applications,” in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, Oct. 2020, pp. 1–9, iSSN: 2155-7209.

[55] H. Helmke, M. Kleinert, S. Shetty, O. Ohneiser, H. Ehr, H. Arilíusson, T. S. Simiganoschi, A. Prasad, P. Motlicek, and K. Veselý, “Readback error detection by automatic speech recognition to increase ATM safety,” 2021, pp. 20–23.

[56] M. Kleinert, H. Helmke, S. Shetty, O. Ohneiser, H. Ehr, A. Prasad, P. Motlicek, and J. Harfmann, “Automated Interpretation of Air Traffic Control Communication: The Journey from Spoken Words to a Deeper Understanding of the Meaning,” in *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*, Oct. 2021, pp. 1–9, iSSN: 2155-7209.

[57] Y. Shi, G. Ma, J. Ren, H. Zhang, and J. Yang, “An End-to-End Conformer-Based Speech Recognition Model for Mandarin Radiotelephony Communications in Civil Aviation,” in *Biometric Recognition*, ser. Lecture Notes in Computer Science, W. Deng, J. Feng, D. Huang, M. Kan, Z. Sun, F. Zheng, W. Wang, and Z. He, Eds. Cham: Springer Nature Switzerland, 2022, pp. 335–347.

[58] Z. Xiao, G. Jia, and B. Shi, “Speech Recognition Model of Civil Aviation Radiotelephony Communication Based on Improved Conformer,” in *2022 International Conference on Automation, Robotics and Computer Engineering (ICARCE)*, Dec. 2022, pp. 1–6.

[59] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer,” Jul. 2020, arXiv:1910.10683 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1910.10683

[60] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. Dallas Texas USA: ACM, May 2000, pp. 93–104. [Online]. Available: https://dl.acm.org/doi/10.1145/342009.335388

[61] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, and V. Dubourg, "Scikit-learn: Machine learning in Python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011, publisher: JMLR. org.

[62] M. Sahidullah and G. Saha, "Design, analysis and experimental evaluation of block based transformation in MFCC computation for speaker recognition," *Speech Communication*, vol. 54, no. 4, pp. 543–565, May 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167639311001622

[63] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*, ser. ICML '06. New York, NY, USA: Association for Computing Machinery, Jun. 2006, pp. 369–376. [Online]. Available: https://dl.acm.org/doi/10.1145/1143844.1143891

[64] M. J. Hunt, "Figures of merit for assessing connected-word recognisers," *Speech Communication*, vol. 9, no. 4, pp. 329–336, Aug. 1990. [Online]. Available: https://www.sciencedirect.com/science/article/pii/016763939090008W

[65] B. Ginsburg, P. Castonguay, O. Hrinchuk, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, H. Nguyen, Y. Zhang, and J. M. Cohen, "Stochastic Gradient Methods with Layer-wise Adaptive Moments for Training of Deep Networks," May 2019. [Online]. Available: https://arxiv.org/abs/1905.11286v3

[66] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019.

[67] O. Kuchaiev, J. Li, H. Nguyen, O. Hrinchuk, R. Leary, B. Ginsburg, S. Kriman, S. Beliaev, V. Lavrukhin, J. Cook, P. Castonguay, M. Popova, J. Huang, and J. M. Cohen, "NeMo: a toolkit for building AI applications using Neural Modules," Sep. 2019, arXiv:1909.09577 [cs, eess]. [Online]. Available: http://arxiv.org/abs/1909.09577

[68] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, "Transformers: State-of-the-Art Natural Language Processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Q. Liu and D. Schlangen, Eds. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://aclanthology.org/2020.emnlp-demos.6

[69] P. Gage, "FEB94 A New Algorithm for Data Compression," 1994. [Online]. Available: http://www.pennelynn.com/Documents/CUJ/HTML/94HTML/19940045.HTM

[70] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[71] J. Burkardt, "The truncated normal distribution," *Department of Scientific Computing Website, Florida State University*, vol. 1, p. 35, 2014.

[72] I. Loshchilov and F. Hutter, “Decoupled Weight Decay Regularization,” Jan. 2019, arXiv:1711.05101 [cs, math]. [Online]. Available: http://arxiv.org/abs/1711.05101

[73] W. A. Falcon, “Pytorch lightning,” *GitHub*, vol. 3, 2019.